

AD-A116 070

GEORGIA INST OF TECH ATLANTA ENGINEERING EXPERIMENT --ETC F76 972
INVESTIGATE CAPABILITY OF ADA HIGHER ORDER PROGRAMMING LANGUAGE--ETC(U)
MAR 82 L J GALLAHER F30602-78-C-0120

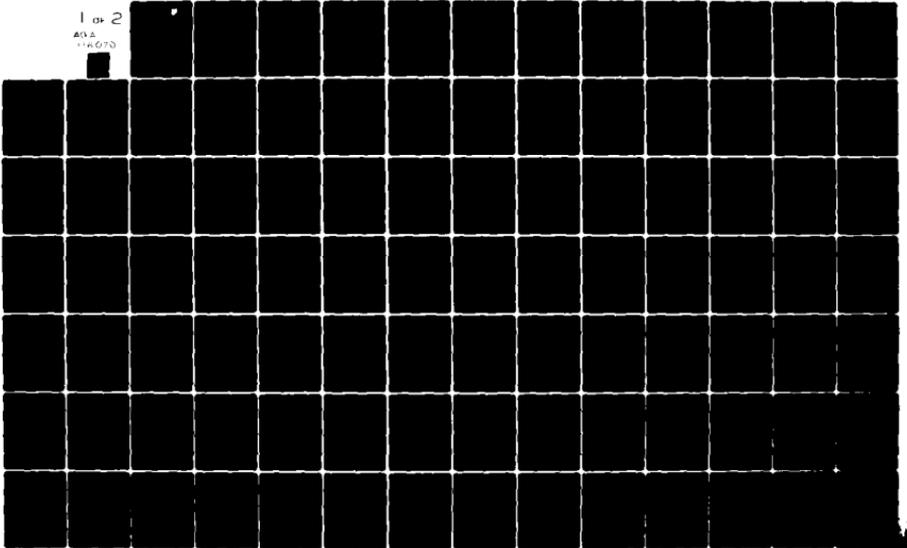
UNCLASSIFIED

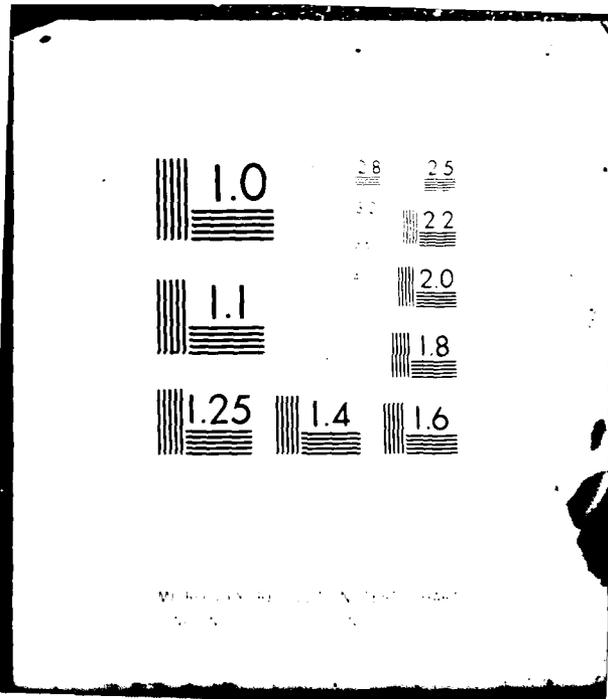
RADC-TR-82-46

NL

1 of 2

ADA
F30602





Resolution Test Chart
1963

12

AD A116070

RADC-TR-82-46
Final Technical Report
March 1982



INVESTIGATE CAPABILITY OF ADA HIGHER ORDER PROGRAMMING LANGUAGE FOR DEVELOPING MACHINE INDEPENDENT SOFTWARE

Georgia Institute of Technology

L.J. Gallaher

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC
ELECTE
JUN 25 1982
S D
E

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

DTIC FILE COPY

82 00 20 006

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-82-46 has been reviewed and is approved for publication.

APPROVED: *Elizabeth S. Kean*

ELIZABETH S. KEAN
Project Engineer

APPROVED:

John J. Marciniak

JOHN J. MARCINIAK, Colonel, USAF
Chief, Command and Control Division

FOR THE COMMANDER:

John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1 REPORT NUMBER RADC-TR-82-46	2 GOVT ACCESSION NO.	3 SECURITY CLASSIFICATION NUMBER
4 TITLE (and Subtitle) INVESTIGATE CAPABILITY OF ADA HIGHER ORDER PROGRAMMING LANGUAGE FOR DEVELOPING MACHINE INDEPENDENT SOFTWARE		6 TYPE OF REPORT & PERIOD COVERED Final Technical Report
7 AUTHOR(s) L.J. Callaher		8 PERFORMING ORG. REPORT NUMBER N/A
9 PERFORMING ORGANIZATION NAME AND ADDRESS Georgia Institute of Technology Engineering Experiment Station Atlanta GA 30332		5 CONTRACT OR GRANT NUMBER(s) F30602-78-C-0120
11 CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (COES) Griffiss AFB NY 13441		10 PROGRAM ELEMENT, SUBJECT TERM, AND REPORT NUMBER 62702F 55811920
12 DISTRIBUTION STATEMENT (if different from Controlling Office) Same		13 REPORT DATE March 1982
		14 SECURITY CLASS. OF THIS REPORT UNCLASSIFIED
		15 SECURITY CLASS. OF THIS REPORT UNCLASSIFIED
16 DISTRIBUTION STATEMENT (if different from Controlling Office) Approved for public release; distribution unlimited		
17 DISTRIBUTION STATEMENT (if different from Controlling Office) Same		
18 SUPPLEMENTARY NOTES RADC Project Engineer: Elizabeth S. Kean (COES)		
19 KEY WORDS (Continue on reverse side if necessary and identify by block number) Ada TAN machine independent HOL COTAN Chebyshev SIN LOG Polynomial COS MATH		
20 ABSTRACT (Continue on reverse side if necessary and identify by block number) In this investigation of the ability of Ada to support machine independent software, a library package of the elementary mathematical functions (sin, cos, en, etc.) was implemented and tested on the Ada/ED Compiler Version 11.4. The Ada language constructs proved quite useful and effective in creating the math function package. The programs were written and successfully syntax checked; however, flaws in this version of the compiler prevented a thorough debugging of these routines. (over)		

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

The routines were designed to be machine and accuracy independent. Accuracy independence was obtained using variable length polynomials whose coefficients are computed (at compile time) from Chebyshev series. For increased efficiency, the normally machine dependent operations (bit picking) are isolated into subroutines that can be optimized for individual installations and hardware.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT

In this investigation of the ability of Ada to support machine independent software, a library package of the elementary mathematical functions (sin, cos, ln, etc.) was implemented and tested on the Ada/ED Compiler Version 11.4. The Ada language constructs proved quite useful and effective in creating the math function package. The programs were written and successfully syntax checked; however flaws in this version of the compiler prevented a thorough debugging of these routines.

The routines themselves were designed to be both machine and accuracy independent; that is the accuracy can be specified at the time the users integrate the package into their program. Accuracy independence was obtained using variable length polynomials whose coefficients are computed (at compile time) from Chebyshev series. For increased efficiency the normally machine dependent operations ("bit-picking") are isolated into subroutines that can be optimized for individual installations and hardware.

An appendix lists the library package.

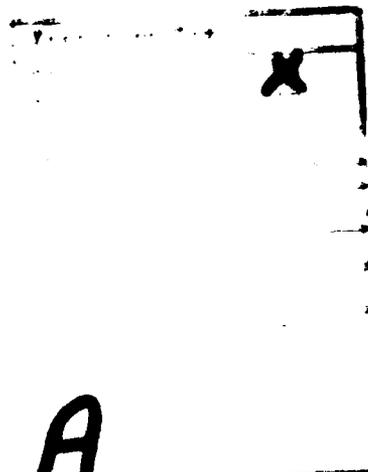


TABLE OF CONTENTS

	<u>Page</u>
I. Introduction	1
II. Theory	2
A. Range Reduction	7
B. Chebyshev Approximation	7
C. Other Approximations	9
III. Ass Considerations	10
IV. The Routine	11
A. Introduction	11
1. Variable Accuracy	11
2. Questions of Efficiency	12
3. Error Tolerances	12
4. Error Conditions and Out-of-Range Alarms	17
B. The Functions	21
1. SIN and COS	21
2. TAN	21
3. ATAN	21
4. ASIN	20
5. ACOS	22
6. EXP	20
7. LN	10
8. SQRT	21
9. SINH and COSH	21
10. ERFC	22
11. ASINH	22
12. ACOSH	22
13. ATANH	22
V. CRITIQUE	26
A. What Went Wrong	26
B. What Went Right	26
C. Unresolved Problems	27
VI. Results and Conclusions	66
VII. Recommendations for Further Study	66
List of References	66
Appendix A	
Appendix B	

LIST OF TABLES

	Page
Table II A-1	6
Table IV A. 1-1	13
Table IV A. 1-II	16
Table IV A. 2-1	17
Table IV A. 2-II	20
Table IV A. 4-1	25
Table IV C-1	31
Table V A-1	38

LIST OF FIGURES

	Page
Figure IV A. 4-1	25
Figure V A-1	41
Figure V B-1	42

Chebyshev derived power series approximations, this will mean fewer terms in the power series. Since accurate requirements are known at compile time, using derivatives, it is possible to determine both the number of Chebyshev terms and the values of the coefficients of the power series at that point.

In the case of the exponential, square root and logarithmic functions, polynomial approximations are used after mapping the function into some relatively small interval, calculated with a power of 2. This kind of mapping is generally done by subtracting a power of 2 from the argument, a process that usually does not require a multiplier table. Since this can be done in a multiplier table, it is possible to use a multiplier table to do the mapping. This approach is used in the IBM 7090 and 7095 and in the IBM 704. In each machine the multiplier table is used to calculate the power of 2 to subtract from the argument of the function. The multiplier table is used to calculate the power of 2 to subtract from the argument of the function. The multiplier table is used to calculate the power of 2 to subtract from the argument of the function.

In the case of the exponential function, the argument is mapped into a small interval and the function is approximated by a power series. The multiplier table is used to calculate the power of 2 to subtract from the argument of the function. The multiplier table is used to calculate the power of 2 to subtract from the argument of the function.

The multiplier table is used to calculate the power of 2 to subtract from the argument of the function. The multiplier table is used to calculate the power of 2 to subtract from the argument of the function.

3. Theory

The multiplier table is used to calculate the power of 2 to subtract from the argument of the function. The multiplier table is used to calculate the power of 2 to subtract from the argument of the function.

for the elementary math functions; during the late fifties and early sixties it held a unique position in almost every computing center library. As demand for greater accuracy increased, improved approximations began to appear in the literature until in 1966 "Approximations for Digital Computers" by Hart, et al. superseded Hastings as the standard reference. This monumental work brought together the best techniques then available and supplied tables of coefficients for several thousand different approximations. The work done here lends heavily on the work of Hart and company as the only references available; however this dependency is mainly on philosophy of approach. All of the actual (Chebyshev) coefficients used in these programs were calculated separately and not taken from Hart. This was done necessary by the fact that the form of the coefficients in Hart, as coefficients of the power series polynomial, is not the form needed here; the form needed here is the coefficients of the Chebyshev series. But since nearly all of the approximations here are equivalent to one of the other forms given in Hart, we could check on and use the corresponding other coefficients given there, and these are cited where they are expected to compare.

Hart and company give a large number of approximations for the various elementary functions in the form $P_n(x)/Q_n(x)$ where P and Q are polynomials. The approximations were here not in this form but with the restriction $Q(x) = 1$; that is, we used just single single polynomial approximations, not the ratio of two polynomials. There were several reasons for this. First, it is well known that the P/Q approximations are significantly better (i.e., faster for the same accuracy) than the corresponding single polynomial with the same number of total coefficients available. Much depends on the nature of the divide operation -- usually the fact on how close to the floating point divide operation relative to the difficulty

operation. In 1968 it was assumed that the hardware speed ratio for multiply/divide was about 12/13. Today this ratio is more typically 1/2 to 1/4 with multiply gaining in speed relative to divide as newer models appear. In fact some of the array processors have no divide at all. Under these circumstances it seems slightly questionable as to whether $P_{12}(z)$ approximations are significantly faster than the simpler single polynomial.

Other reasons for restricting to a single polynomial is that it is the most simple -- to find the near optimal polynomials for a given accuracy -- the Chebyshev approximations do this -- and the number of constants that need to be stored is then relatively small.

11. A Further Reduction

There are many ways to improve the efficiency in approximating a function. One of the more powerful of these is to reduce the range over which the approximation is done. This is done by partitioning the range into sub-ranges and approximating the function separately on each sub-range. This is done for example in the polynomial approximation of the cosine by the P_{12} polynomials. One can further reduce the range of the approximation to the interval $[-\pi/4, \pi/4]$ and use the interval $[\pi/4, \pi/2]$ as well.

$$c(x) = c_1(x) + c_2(x) + \dots + c_n(x)$$

$$c_1(x) = c_1(x) + \dots + c_n(x)$$

and finally one can reduce the range to $[-\pi/8, \pi/8]$ and use the interval $[\pi/8, \pi/4]$ as well.

$$c(x) = c_1(x) + c_2(x) + \dots + c_n(x)$$

$$c_1(x) = c_1(x) + \dots + c_n(x)$$

One can also use the fact that the cosine function is even to reduce the range of the approximation and the cosine from the sine approximation. A further reduction in the interval $[-\pi/8, \pi/8]$ can be achieved by using use of symmetry. However the symmetry is usually exploited

by the use of even or odd polynomials, i.e.,

$$\sin(x) \approx P_s(x^2)$$

$$\cos(x) \approx P_c(x^2)$$

where P_s and P_c are the sine and cosine polynomial approximations.

Table II.A-1 gives the ranges over which the various functions are approximated by polynomials. The various range reductions relations used are listed in the program. (See Appendix A.)

In doing range reduction for the functions EXP, LN and SQRT, it is assumed that the computer hardware in use is basically binary. (This is consistent with both the proposed IEEE standards and MCF or Nebula specifications.) For each of the functions LN and SQRT the argument is separated into a characteristic and an exponent of 2, i.e.,

$$x = 2^J p$$

where J is integer and $0.5 \leq p < 1.0$.

then for SQRT

$$\text{SQRT}(x) = \text{SQRT}(2^J p) = 2^{J/2} \sqrt{p},$$

and \sqrt{p} is approximated by a polynomial.

For LN

$$\text{LN}(x) = \text{LN}(2^J p) = J \ln 2 + \ln p,$$

and $\ln p$ is approximated by a polynomial.

In the case of the EXP function the argument is first mapped onto a power of 2 by

$$x^* = x \ln 2,$$

and x^* broken down into an integer and remainder part $x^* = k + p$ where k is integer and $-0.5 \leq p \leq 0.5$ so that

$$e^{x^*} = 2^{x^* / \ln 2} = 2^k \cdot 2^{x^* p} = 2^k 2^p.$$

Here $2^{(k)}$ is approximated by a polynomial and if $p \leq 0$ the reciprocal is

TABLE II.A-1

Ranges over which polynomial approximations are used
for the various functions.

Function	Interval
SIN	$\pm \pi/4$,
COS	$\pm \pi/4$
TAN	$\pm \pi/8$
ATAN	$\pm(\sqrt{2} - 1)$
ASIN	± 0.375
EXP	0, 0.5
LN	0.5, 1.0
SQRT	0.5, 1.0
SINH	± 1.0
COSH	± 1.0
TANH	± 0.5
ASINH	± 0.375
ATANH	± 0.25
ACOS	polynomial not used
ACOSH	polynomial not used

taken. The operations dealing with the powers of 2 are assumed to be very fast; they will be machine dependent and should be tailored specifically for each machine.

The hyperbolic functions and their inverses are approximated by polynomials in a narrow region around the origin and outside this region are computed from their relationship to EXP, LN and SQRT.

The inverse functions arccos(x) and invcosh(x) have an anomalous behavior in the neighborhood of x = 1; they cannot be approximated by a polynomial in x near x = 1. Instead we have

$$\arccos(x) \approx \sqrt{2y + y^2/3 + 4y^3/45 + \dots}$$

where y = 1-x, and

$$\operatorname{invcosh}(x) \approx \sqrt{2y - y^2/3 + y^3/45 - \dots}$$

where y = x-1.

The first two terms are used to approximate the functions, and the third term is used to control the range over which the approximation is used in such a way as to maintain the error tolerance. Outside this range the arccosine is computed from the arcsine; the invcosh is computed from LN and SQRT.

For the arcsine, an odd power series is used in the region about the origin; outside this region the arcsine is computed from its relation to arctangent and SQRT function, i.e.,

$$\arcsine(x) = \arctan(x / \sqrt{1-x^2}).$$

II. B. Chebyshev Approximations

The advantage of the Chebyshev polynomials is that they are the polynomials having the largest number of maxima and minima on the given interval and for which all maxima and minima are equal in magnitude. Thus, if the lowest order term neglected can be considered the error function, it can be seen that the error is more or less uniformly distributed over the interval,

and can be shown to have the smallest maximum error of any polynomial approximation of the same order.

Since accuracy requirements are known at compile time, it is possible in Ada to determine both the number of Chebyshev terms and the values of the coefficients to the power series at that point.

Example: $\sin(\pi x)$.

This function can be mapped onto the interval $-1 \leq x \leq +1$. The function can then be well approximated in this range by the n-term Chebyshev series

$$\sin(\pi x) \equiv \sum_{1 \leq k \leq n} B_k T_k(x) \quad -1 \leq x \leq +1$$

where the Bs can be calculated from

$$B_k = \frac{2}{\pi} \int_{-1}^1 T_k(x) \sin(\pi x) dx / \sqrt{1-x^2}.$$

(Only the odd numbered Bs will be nonzero.)

Each $T_k(x)$ is a polynomial of order k

$$T_k(x) = \sum_{0 \leq j \leq k} C_{kj} x^j.$$

The Cs are well known and calculated exactly.

The n-term power series approximation for the sine then is

$$\begin{aligned}\sin(\pi x) &\approx \sum_{1 \leq k \leq n} \sum_{0 \leq j \leq k} B_k C_{kj} x^j \\ &\approx \sum_{0 \leq j \leq n} A_j x^j\end{aligned}$$

where

$$A_j = \sum_{j \leq k \leq n} B_k C_{kj}.$$

We see that adding more Chebyshev terms, say to improve the accuracy, changes all the A coefficients of the power series. Similar considerations apply to approximations for the other functions.

II.C. Other Approximations

In addition to Chebyshev derived polynomials, two other approximations commonly used are Pade' like approximations and continued fraction. The Pade' like method consists of approximating as a function a ratio of two polynomials $P(x)/Q(x)$; it can be converted to an equivalent continue fraction and vice versa. While these methods can sometimes be superior to a simple linear Chebyshev polynomial approximation in the sense that the same or better accuracy can be obtained with fewer computer operations, they all suffer the same drawbacks from the point of view of trying to write machine and accuracy independent programs. This is that for optimum conditions, all coefficients or constants in the method will change if the accuracy is changed. The Pade' and continued fraction methods suffer other problems.

First, it is much more difficult to find the optimum set of coefficients for a given accuracy, while this is relatively easy to do for the Chebyshev series. The Chebyshev series requires storing only a single set of constants

(10) for each function while for the $P(x)/Q(x)$ method, an entire set of constants would be needed for each accuracy interval.

A second reason for not going with the ratio of polynomials is that it requires a division operation and the trend for hardware today is toward a relatively slow divide operation relative to addition and multiply. Present day hardware suggests we should avoid divisions where practical. (See IV.A.2 for a further discussion of this point.)

In summary, the Chebyshev method was chosen over the $P(x)/Q(x)$, the ratio of two polynomials, because

- 1) Variable accuracy methods are relatively easier to obtain;
- 2) Fewer stored constants are required;
- 3) A division operation is eliminated (or traded for some number of multiplications and additions).

III. Ada Considerations

The principal Ada techniques expected to be most useful in developing machine and accuracy independent functions are the package and generic concepts. The main idea is that the functions will be embedded in a package with a generic parameter, call it REAL, describing the type (number of digits) to be used in the floating point arithmetic. The type REAL or its number of digits is then specified by the user in his main program. Then at the time the package and user program are integrated, the package is instantiated with the appropriate number of digits for REAL.

There are a number of program features that depend on the number of digits in REAL (called in Ada REAL'DIGITS).

First, the number of terms in the Chebyshev series and so the power series is determined by requiring that the Chebyshev terms not used are all smaller than $10^{**(-REAL'DIGITS)}$. Next, this determines the size of the arrays that hold the power series polynomial coefficients. Then, in some cases

(ACOS, ACOSH, and TANH) the range in which a particular approximation is used will be determined by the number of digits in REAL; the computation of these ranges is carried out in the initialization block of the function package.

Certain assumptions concerning the computer arithmetic have been made in writing these packages. We believe these assumptions are, in general, consistent with the Ada view, the IEEE standards, and the MCF or Nebula choice of floating point arithmetic. These assumptions are as follow:

1. There is a maximum floating point number (called in Ada `FLOAT' LARGE`).
2. `-FLOAT' LARGE` exists and is the most negative floating point number.
3. Every floating point number including `FLOAT' SMALL`, but excepting zero, has a reciprocal; the reciprocal of `±FLOAT' LARGE` may be zero, but need not be.
4. Any positive number can be subtracted from `FLOAT' LARGE` and any positive number can be added to `-FLOAT' LARGE` without causing an exception alarm.
5. `FLOAT' LARGE` can be divided by any number greater than or equal in magnitude to 1.0 or multiplied by any number less than or equal in magnitude to 1.0 without causing an exception.
6. Any two floating point numbers (including `±FLOAT' LARGE`) can be compared without causing an overflow or exception.

IV. Routines

IV.A. Introduction

IV.A.1. Variable Accuracy

One of the prime goals of this effort was to construct these subroutines

so as to be able to give variable accuracy. The object is to let the user specify the accuracy to which the computations are to be carried out with the idea that the less accuracy required the less time needed for the computation; the user does not need to pay (time-wise) for accuracy not needed. The variable accuracy is achieved here by using variable length polynomials for the approximations. The number of terms in the polynomial is determined at compilation time (or later at "assembly" time) by use of a generic parameter type REAL. REAL is a user specified type and it is the number of digits specified by the user for the type REAL that determines the accuracy and, consequently, the number of terms in the polynomial approximation for each function.

Once the accuracy is determined the polynomial coefficients are computed from the corresponding Chebyshev coefficients. These Chebyshev coefficients have been precalculated and are stored as part of the program or package containing the function. Enough terms in the Chebyshev series are stored to be able to achieve an accuracy (relative error) of 10^{-16} . It is assumed that this accuracy limitation (10^{-16}) will be adequate for the vast majority of the anticipated Ada applications in embedded systems; extension to higher accuracy would be straightforward. Table IV.A.1-i gives a table of the accuracy versus the number of terms needed for each function. There is, of course, for each function a certain amount of computation necessary to reduce the argument range to within the range of the polynomial, but that is not reflected in the table. (Also not reflected in Table IV.A.1-i is the fact that certain functions call other functions for some ranges of their values rather than using a polynomial.) Table IV.A.1-ii lists those functions that call others and in what ranges.

TABLE 2.1.1-1

The order of the polynomial used for each function is dependent on existing parameters

Function	Parameter	Accuracy			
		10^{-4}	10^{-6}	10^{-10}	10^{-16}
SIN	$\pi/4$	4	3	5	6
CSIN	$\pi/4$	4	4	5	6
TAN	$\pi/4$	2	4	6	8
ATAN	$\pi(\sqrt{2}-1)$	2	5	7	10
ASIN	$\pi/375$	2	4	7	9
EXP	0.5, 1.0	3	5	7	9
LN	0.5, 1.0	1	3	5	7
SOFT	0.5, 1.0	1	3	5	9
SINH	$\pi/0$	2	4	5	6
CSINH	$\pi/0$	2	4	6	7
TANH	$\pi/5$	2	5	7	10
ATANH	$\pi/25$	2	4	6	8
ASINH	$\pi/375$	2	4	7	9
ACOS	polynomial not used				
ACOSH	polynomial not used				

Table 14 A-11

Functions that call other functions and the ranges where this is done. These functions receive 0, 1, or 2 arguments and the ranges given here.

Function	Function Called	Range
ABS(X)	ABS	X > 0
ACOS(X)	ACOS	X > 1
ADD(X, Y)	ADD	X > 0, Y > 0
ASIN(X)	ASIN	X > 1
ATAN(X)	ATAN	X > 0
ATAN2(X, Y)	ATAN2	X > 0, Y > 0
CEILING(X)	CEILING	X > 0
EXP(X)	EXP	X > 0
FLOOR(X)	FLOOR	X > 0
LN(X)	LN	X > 0
LOG(X)	LOG	X > 0
LOG10(X)	LOG10	X > 0
MOD(X, Y)	MOD	X > 0, Y > 0
POW(X, Y)	POW	X > 0, Y > 0
ROUND(X)	ROUND	X > 0
SIN(X)	SIN	X > 0
TAN(X)	TAN	X > 0
TRUNC(X)	TRUNC	X > 0

11.4.2 Qualities of Efficiency

There are two kinds of efficiency to be considered when discussing efficiency. One is the efficiency of the individual and the other is the efficiency of the organization. The efficiency of the individual is the ability to do as much as possible and to do it in as little time as possible. The efficiency of the organization is the ability to do as much as possible and to do it in as little time as possible. The efficiency of the individual is the ability to do as much as possible and to do it in as little time as possible. The efficiency of the organization is the ability to do as much as possible and to do it in as little time as possible.

There is an important feature of the individual efficiency which is the ability to do as much as possible and to do it in as little time as possible. The efficiency of the organization is the ability to do as much as possible and to do it in as little time as possible. The efficiency of the individual is the ability to do as much as possible and to do it in as little time as possible. The efficiency of the organization is the ability to do as much as possible and to do it in as little time as possible.

In order to be efficient, the individual must be able to do as much as possible and to do it in as little time as possible. The efficiency of the organization is the ability to do as much as possible and to do it in as little time as possible. The efficiency of the individual is the ability to do as much as possible and to do it in as little time as possible. The efficiency of the organization is the ability to do as much as possible and to do it in as little time as possible.

Table IV A 2-1

In calculating the average marginal cost for the various functions, it is assumed that either the volume of the output or the volume of the input is held constant over each interval. Also, the average cost is assumed to be the average of the marginal cost and the average cost of the input or output, depending on which is held constant.

Function	Input	Output	Average Cost
1000	↓		1.1000
2000	↓		1.1000
3000	↓		1.1000
4000		↓	1.1000
5000		↓	1.1000
6000		↓	1.1000
7000	↓		1.1000
8000	↓		1.1000
9000	↓		1.1000

1. The first part of the document is a list of names and titles, including the names of the authors and the titles of their works. This section is organized in a structured manner, with names and titles clearly separated.

2. The second part of the document contains a detailed description of the works listed in the first part. This section provides information about the content, scope, and significance of each work, as well as any relevant details about the authors or the publication process.

3. The third part of the document is a list of references or citations, which are used to support the information provided in the previous sections. These references are organized in a clear and concise manner, allowing the reader to easily locate the original sources of the information.

4. The fourth part of the document is a concluding section, which summarizes the main findings and conclusions of the document. This section provides a clear and concise overview of the entire document, highlighting the key points and the overall significance of the work.

example, Floating Point Systems' AP-120B, a very high speed auxiliary array processor, doesn't even have a divide; addition and multiplication have the same speed, giving ratios of 1:1:100 for its arithmetic speeds. Again this means that algorithms involving division will be at a significant disadvantage.

In any event, this suggests that one needs to make some kind of assumption concerning relative speeds of arithmetic operations in order to compare different algorithms. As a first approximation the speed ratios in Table IV & V are used to compare the efficiency of algorithms.

It is also clear from the observation of how slow divide is relative to multiply that one should be looking for algorithms that use as few divides as possible. This is one of the justifications for avoiding the $P(x)/Q(x)$, the ratio of two polynomials, algorithms and using the simpler Chebyshev polynomial instead.

IV 4 7 Error Tolerances

One of the main goals here is to achieve an error tolerance consistent with the accuracy specified by the user. The user specifies the number of digits to be used for his particular type of floating point numbers or reals and this then determines the acceptable error. Thus for example, if the user specifies:

type REAL is digit 5.

the acceptable relative error tolerance will be taken as 10^{-5} . This implies then that the power series approximation will be composed of all Chebyshev terms whose absolute value is greater than or equal to 10^{-5} . This computation of the power series coefficients from the Chebyshev coefficients is done in the package initialization. It is also at this point that the size of the power series arrays is determined.

Table 11.4.11

Assumed Relative Speeds for the Arithmetic Operations

Operation	Relative Speed
Add two real numbers	1
Multiply two reals or integers	1
Divide two reals or integers	10
Integer add or compare	100000
Logical operations	1
Multiplication and division by 2	1
Compare two real numbers	1

There is one exception to the above description of how the error is controlled and that is for the SQRT function. In the square root function a single Newton-Raphson or Heron's iteration is performed; i.e.,

$$x = (y/x + x)/2$$

where $x = \sqrt{y}$. This operation squares the relative error when the error is small. For SQRT it is only the first approximation that is obtained from the Chebyshev polynomials, the final value is obtained from the above iteration. Thus the accuracy of the power series need be only as good as the square root of the relative error tolerance; i.e., if the relative error tolerance is 10^{-5} , the polynomial need only be as accurate as $10^{-2.5} = 3.16^{-5}$ -- the Heron iteration will improve this back to 10^{-5} .

The Chebyshev approximations normally give absolute not relative error limits. When dealing with floating point arithmetic it is relative error that one wishes to maintain more or less uniform over some interval. For the even functions this is not a problem since in the interval of approximation the functions do not change greatly and so the relative error is very nearly equal to the absolute error (the functions being approximately 1 at the origin). To maintain the relative error more or less constant for the odd functions, say $f(x)$, we use a Chebyshev approximation to $f(x)/x$. This gives nearly uniform relative error for $f(x)$ on the interval around $x = 0$, and again the absolute error is approximately equal to the relative since for these odd functions $\lim_{(x \rightarrow 0)} f(x)/x = 1$. So for example, the Chebyshev coefficients for the SIN function are actually those for $\sin(x)/x$, and those for TAN are in fact for $\tan(x)/x$, etc., for all the odd functions.

IV.A.4. Error Conditions and Out of Range Alarms

In nearly all computer arithmetic systems there is a limit to the size of a floating point number; there is also a smallest-in-magnitude nonzero number. These numbers are machine dependent but referable in Ada as F'LARGE and F'SMALL, where F refers to the (floating point) type. Because of these limits there are certain functions such as EXP, SINH, COSH, etc. for which only a relatively restricted range of inputs are valid. Thus for example, if we refer to our floating point type as REAL, input values to the EXP function greater than $\ln(\text{REAL}'\text{LARGE})$ cannot be computed since they would result in values greater than REAL'LARGE. There are other functions such as LN and SQRT that have invalid ranges because the result would not be in the real number system (but would be complex). For example, numbers less than or equal to zero are invalid inputs for LN; $X < 0$ is invalid for SQRT(X).

The problem of what to do about invalid inputs is complicated by the fact that all computer arithmetic with floating point numbers involves rounding; floating point arithmetic is seldom exact. When rounding takes place in the neighborhood of the boundary between the valid and invalid region of the function, it is difficult to decide what is the best action to take. Thus for example, if one wants to evaluate:

$$\text{SQRT}(A + B + C - \text{SIN}(D))$$

and the exact value of $A+B+C-\text{SIN}(D)$ is 0 but due to rounding (or truncation in SIN) the computer results turns out to be $-1.0 \text{ E}-9$, what is the proper action? How should the SQRT subroutine respond to small negative inputs, and what is small? One action would be to trip a numeric exception alarm and let (force) the user handle the problem. Some systems halt the computation at this point.

Another solution is to assume that there is a small negative region such that input from that region ought really be treated as input of zero and not trip the exception alarm for these inputs, simply return zero; however it is difficult to determine how large or small this boundary interval should be.

The solution to the problem of the invalid input here is different yet from those mentioned above. We take the point of view that there should be no truly invalid input region and redefine our functions so all inputs are legitimate and will return some value and no exceptions are ever raised. For example, SQRT is defined so that

$$\text{SQRT}(X) = \begin{cases} \sqrt{X} & X \geq 0, \\ 0 & X < 0 \end{cases}$$

The square root function given here always returns a value, with no exception raised for input values from -REAL'LARGE to REAL'LARGE. Thus, small rounding errors in the vicinity of zero will cause no difficulties; however, it is the users responsibility to trap any input values to SQRT that the user feels ought to be considered illegitimate. Table IV.A.4-I gives the extended definition of the elementary functions showing how the usually considered invalid regions are treated. As an example, Figure IV.A.4-I shows how the arcsine function is extended beyond the normal input range of ± 1.0 .

As mentioned above the main virtue of extending the definitions of the functions into their normally invalid regions is to avoid extraneous error signals when the argument drifts into the invalid region by a small amount of rounding or truncation error. This solution may not be the best resolution to the problem nor the solution desired by a particular user. It does however have the virtue of allowing the user to determine what he would like to see as

Figure IV.A.4-1
The extended definition
of the arcsine function.

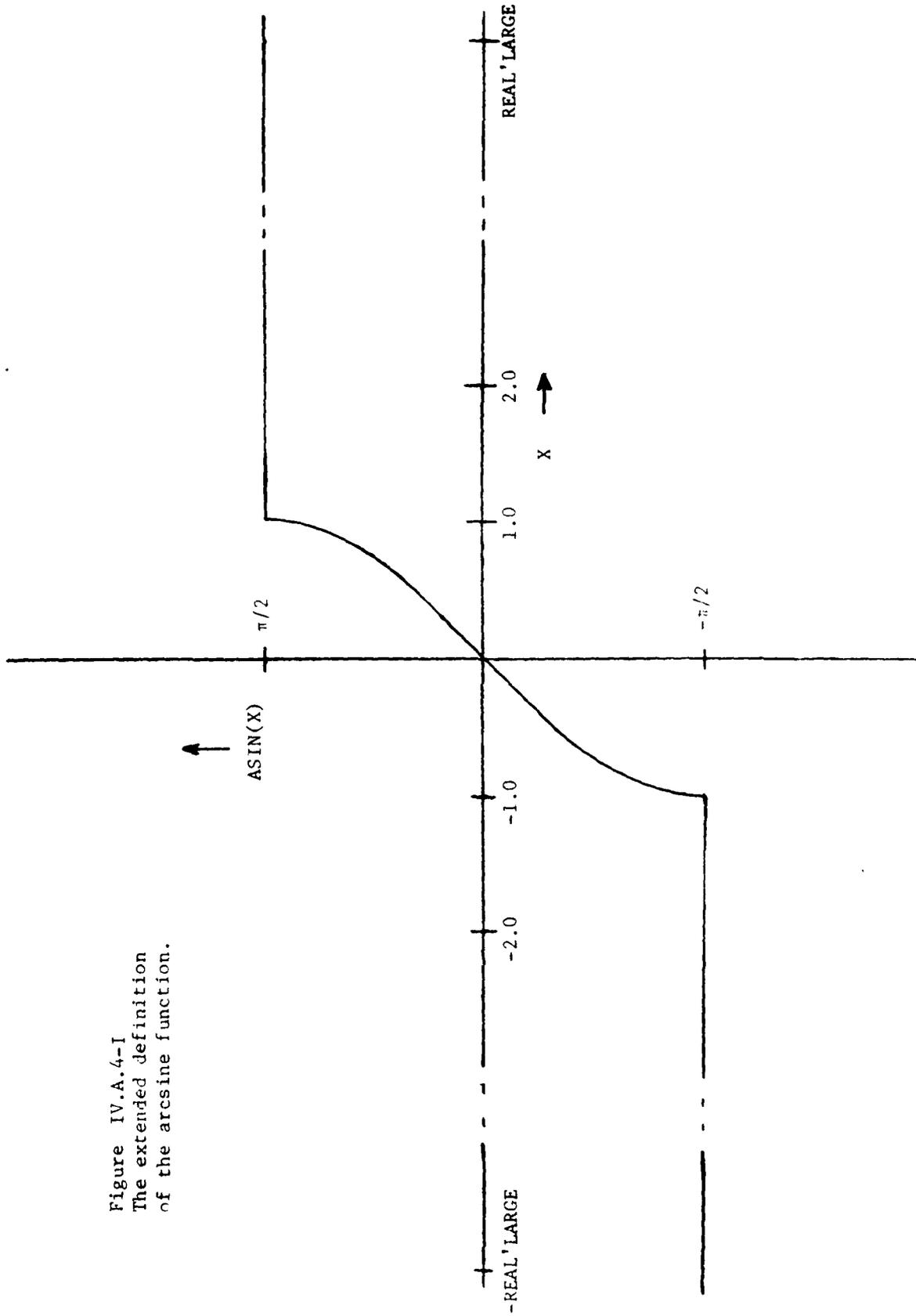


Table IV.A.4-I

A list of the functions and the way in which their inputs were extended to the normally invalid regions, together with the output values in these regions.

SIN(x) = sin(x)	all x
COS(x) = cos(x)	all x
TAN(x) = tan(x)	all x
	(returns \pm REAL'LARGE
	for x = $\pm \pi/2, \pm 3\pi/2, \pm 5\pi/2 \dots$)
ASIN(x) =	$\begin{cases} \arcsin(x) & x \leq 1.0 \\ + \pi/2 & x > 1.0 \\ - \pi/2 & x < -1.0 \end{cases}$
ACOS(x) =	$\begin{cases} \arccos(x) & x \leq 1.0 \\ 0 & x > 1.0 \\ -\pi & x < -1.0 \end{cases}$
ATAN(x) = arctan(x)	all x
EXP(x) =	$\begin{cases} e^x & x \leq \ln(\text{REAL'LARGE}) \\ 0 & x < -\ln(\text{REAL'LARGE}) \\ \text{REAL'LARGE} & x > \ln(\text{REAL'LARGE}) \end{cases}$
LN(x) =	$\begin{cases} \ln(x) & x > 0 \\ -\text{REAL'LARGE} & x \leq 0 \end{cases}$
SQRT(x) =	$\begin{cases} \sqrt{x} & x \geq 0 \\ 0 & x < 0 \end{cases}$
SINH(x) =	$\begin{cases} \sinh(x) & x \leq \ln(\text{REAL'LARGE}) \\ \text{REAL'LARGE}/2 & x > \ln(\text{REAL'LARGE}) \\ -\text{REAL'LARGE}/2 & x < -\ln(\text{REAL'LARGE}) \end{cases}$

Table IV.A.4-1
(Cont'd)

TANH(x) = tanh(x)	all x
ASINH(x) = invsinh(x)	all x
ACOSH(x) = {	
invcosh(x)	x > 1.0
0	x < 1.0
ATANH(x) = {	
invtanh(x)	x < 1.0
REAL'LARGE	x > 1.0
-REAL'LARGE	x < -1.0

the valid regions and allows (requires) him to set his own error traps for what that user considers illegitimate inputs.

IV.B.1. SIN and COS

Input Value Range: -REAL'LARGE to REAL'LARGE (in radians)

Output Value Range: -1.0 to +1.0

The argument, x , is first mapped onto the interval $-3\pi/2$ to $3\pi/2$. If $x \leq \pi/4$ the sine or cosine polynomial is computed for the respective function; if $|x|$ is greater than $\pi/4$, SIN calls the cosine polynomial and COS calls the sine polynomial with the appropriate sign and phase adjustment.

There are no invalid input or output ranges for SIN and COS. However because of truncation and/or rounding errors it is possible for results to be returned which are slightly larger in magnitude than 1.0.

IV.B.2. TAN

Input Value Range: -REAL'LARGE to REAL'LARGE (in radians)

Output Range: -REAL'LARGE to REAL'LARGE

The argument, x , is first mapped onto the interval $-5\pi/8$ to $5\pi/8$. If $|x|$ then is less than $\pi/8$ the tangent polynomial is called; if $|x|$ is in the range $\pi/8$ to $3\pi/8$ the tangent polynomial for $|x| - \pi/4$ is computed (as Q) and TAN is evaluated as $(Q+1)/(1-Q)$, with appropriate sign[†]. If $|x| > 3\pi/8$, TAN is computed from the reciprocal of the tangent polynomial with appropriate sign and phase adjustment.

The only invalid input arguments are $|x| = n\pi/2$, $n=1, 3, 5$, etc. For these angles the value \pm REAL'LARGE is returned. No error halt or exceptions are generated.

[†] This is an application of the relation :

$$\tan x = (\tan(x-b) + \tan b)/(1 - \tan(x-b)\tan b)$$

with $b = \pm \pi/4$

IV.B.3. ATAN

Input Value Range: -REAL*LARGE to REAL*LARGE

Output Range: - $\pi/2$ to $\pi/2$ (in radians)

If the argument, x , is in the range $|x| \leq \sqrt{3} = 1.0$, the arctan polynomial approximation is used. If $|x| > \sqrt{3} = 1.0$, then arctangent polynomial is used with $1/x$ as argument with appropriate sign and phase adjustment. In the intermediate range, that is $\sqrt{3} = 1.0 < |x| < \sqrt{3} = 1.0$, the arctan polynomial is used with argument $(|x| - 1.0)/(|x| + 1.0)$ and appropriate phase and sign adjustment.

There is no invalid input range.

IV.B.4. ASIN

Valid Input Value Range: -1.0 to +1.0

Output Range: - $\pi/2$ to $\pi/2$ (in radians)

If the argument, x , is in the region $|x| \leq 0.175$ then the arcsine polynomial approximation is used; if x lies outside this region then the relation:

$$\arcsine x = \arctan(x \sqrt{1.0 - x^2})$$

is used. Thus the arcsine routine uses both the arctan and the square root routine.

If the input is outside the valid region, that is outside -1.0 to 1.0, the value $\pm\pi/2$ is returned depending on the sign of the argument. Note: no error halt nor exception is generated for input values outside the valid region -1.0 to 1.0 (see Section IV.A.4 above).

IV. D. 5. ACOS

Valid Input Value Range: -1 to +1

Output Range: 0 to π radians

If the argument, x , is in the neighborhood of 0, then a special approximation is used as follows:

$$\theta = \sqrt{\frac{2}{5}} x + \frac{1}{5} x^3 + \frac{1}{63} x^5 + \dots$$

where θ is the arcsine of x and $\pi/2$ is used for the special case. This power series for which the error is θ is approximately half the third

term. The relative error is then approximately $\frac{\theta^3}{24x^3} = \frac{1}{24} \frac{x^2}{15}$ where $\theta = \sqrt{2/5} x$

has been used. This relative error needs to be less than 10% (REAL(DIGITS) - 10) so we need

$$\frac{\sqrt{2}}{15} x^{5/2} \leq 10 \times (\text{REAL(DIGITS)} - 10)$$

The range of x then is roughly $x = \pm 1.2 \times 10^{(REAL(DIGITS) - 10) \times 2/5}$ and $0.0 = \sqrt{15^2/2}$.

Outside the above range the arcsine is calculated from $\theta/2 + \arcsin$. If the argument is outside the valid region, that is outside -1 to +1, the value 0 or π is returned depending on the sign of the argument. Note: no error halt nor exception is generated for input values outside the valid range (see Section IV. A. 4 above).

1.1.1. Case

Let f be a function from \mathbb{R}^n to \mathbb{R}^m .

Let S be a subset of \mathbb{R}^n .

The image of S under f is the set $f(S) = \{f(x) \mid x \in S\}$.
The domain of f is \mathbb{R}^n . The range of f is \mathbb{R}^m .
The image of S under f is the set $f(S)$.
The domain of f is \mathbb{R}^n . The range of f is \mathbb{R}^m .

$$f(x) = \begin{pmatrix} x_1^2 + x_2^2 \\ x_1 - x_2 \end{pmatrix}$$

Let $S = \{(x, y) \mid x^2 + y^2 \leq 1\}$.

The image of S under f is the set $f(S)$.
The domain of f is \mathbb{R}^2 . The range of f is \mathbb{R}^2 .
The image of S under f is the set $f(S)$.
The domain of f is \mathbb{R}^2 . The range of f is \mathbb{R}^2 .

1.1.2. Case

Let f be a function from \mathbb{R}^n to \mathbb{R}^m .

Let S be a subset of \mathbb{R}^n .

The image of S under f is the set $f(S)$.
The domain of f is \mathbb{R}^n . The range of f is \mathbb{R}^m .

$$f(x) = \begin{pmatrix} x_1^2 + x_2^2 \\ x_1 - x_2 \end{pmatrix}$$

where I is an integer and $1 \leq I \leq n$. See also 1.1.1. The natural log of I is

computed from the two polynomials $A(x) = a_0 + a_1x + \dots + a_{n-1}x^{n-1}$ and $B(x) = b_0 + b_1x + \dots + b_{m-1}x^{m-1}$

If the argument, x , is not in the range $[-REAL_LARGE, REAL_LARGE]$, then the value $REAL_LARGE$ is returned, and an error flag is generated and control is given to the main program.

IV. B. 8. SINH

Valid Input Value Range: $[-REAL_LARGE, REAL_LARGE]$

Output Range: $[-REAL_LARGE, REAL_LARGE]$

To compute the square root, the argument, x , is first converted into a binary expansion and a fraction part and then

$$x = \epsilon \cdot 2^j$$

where j is integer and $1/2 \leq \epsilon < 1$. The square root of ϵ is approximated by a series and the square root of 2^j is obtained as $2^{j/2}$. The product of these two terms is approximated by the addition of terms of the series and the value of the error flag is within tolerance.

If the argument, x , is not in the value range that is defined by the $REAL_LARGE$ $[-REAL_LARGE, REAL_LARGE]$, then the value $REAL_LARGE$ is returned and an error flag is generated. Note that because of rounding and/or truncation errors $SINH(x)$ will not always equal x and it may be less than $REAL_LARGE$ even if $x < REAL_LARGE$.

Also, as a result of the round operation, the error is $SINH(x)$ is always positive.

IV. B. 9. SINH and COSH

Valid Input Value Range: $[-REAL_LARGE, REAL_LARGE]$

Output Range: $[-REAL_LARGE, REAL_LARGE]$

The first part of the proof, ϕ , is the same as the first part of the previous proof. The second part, ψ , is the same as the second part of the previous proof. The third part, χ , is the same as the third part of the previous proof. The fourth part, η , is the same as the fourth part of the previous proof.

$$\begin{aligned}
 \text{part } \phi &= x^2 + y^2 + z^2 \\
 \text{part } \psi &= x^2 + y^2 + z^2
 \end{aligned}$$

The first part of the proof, ϕ , is the same as the first part of the previous proof. The second part, ψ , is the same as the second part of the previous proof. The third part, χ , is the same as the third part of the previous proof. The fourth part, η , is the same as the fourth part of the previous proof.

THE END

The first part of the proof is the same as the first part of the previous proof. The second part is the same as the second part of the previous proof. The third part is the same as the third part of the previous proof. The fourth part is the same as the fourth part of the previous proof.

The first part of the proof, ϕ , is the same as the first part of the previous proof. The second part, ψ , is the same as the second part of the previous proof. The third part, χ , is the same as the third part of the previous proof. The fourth part, η , is the same as the fourth part of the previous proof.

$$\begin{aligned}
 \text{part } \phi &= \frac{x^2 + y^2 + z^2}{x^2 + y^2 + z^2} \\
 \text{part } \psi &= \frac{x^2 + y^2 + z^2}{x^2 + y^2 + z^2}
 \end{aligned}$$

Note that all inputs of A are valid. No error halts or exceptions are generated by A .

2.1.3 ACOSH

Valid Input Value Range: $REAL_MIN$ to $REAL_MAX$
Output Range: 0 to $REAL_MAX$

If the input argument, x , is in the interval 0.999 to 0.9999 , then the hyperbolic cosine is approximated with a polynomial.
If x is in the interval $REAL_MIN$ to $REAL_MAX$, then the $ACOSH$ is computed from

$$ACOSH(x) = \sqrt{x^2 - 1}$$

with appropriate sign. If x is in the interval $REAL_MIN$ to $REAL_MAX$, $ACOSH$ is computed from
2.1.3.1 with appropriate sign.

Note that all inputs are valid and will return valid outputs. However, there will be values of x in the interval $REAL_MIN$ to $REAL_MAX$ for which

$ACOSH(x)$ is 0 and the inaccuracy will be quite large. Also the relationship

$$ACOSH(ACOSH(x)) = x$$
 should fail badly for x values such that
 x is in the interval $REAL_MIN$ to $REAL_MAX$.

2.1.3.2 ASINH

Valid Input Value Range: $REAL_MIN$ to $REAL_MAX$
Output Range: 0 to $REAL_MAX$

If the argument, x , is in the neighborhood of 1.0 , then a special approximation is used, as follows

$$ASINH(x) = \sqrt{2x - x^2} - \frac{1}{2} \ln(1 + x)$$

where θ is the inverse hyperbolic cosine and $y = x - 1.0$. By using the first two terms in this power series, for small y , the error in θ is approximately half the third term; the relative error is then approximately

$$\frac{y^3}{2048} = \frac{y^3}{2^11}$$

where $\theta = \sqrt{y}$ has been used. This relative error needs to be held below $1000 \cdot \text{REAL}(\text{LARGE})$, so we need

$$\frac{y^3}{2^11} < 1000 \cdot \text{REAL}(\text{LARGE})$$

on the range of y then is roughly

$$y < \sqrt[3]{2048 \cdot 1000 \cdot \text{REAL}(\text{LARGE})} \quad \text{here } 2^11 = 2048 \text{ and } 4000 = \sqrt[3]{2048 \cdot 1000}$$

Outside of the above range the approximation is divided into two regions. For $x < \sqrt{\text{REAL}(\text{LARGE})}$, the approximation $\ln(\sqrt{x^2-1} + x)$ is used and above this INCOSH is used for ACOSH .

For small inputs, that is for $x \approx 0$, the value zero is returned, no error halt or exception is generated (see Section IV.A.4 above).

There will be values of x ($x = \text{REAL}(\text{LARGE})/2$) for which the relation $\text{COSH}(\text{ACOSH } x) = x$ will fail and the discrepancy will be quite large.

Also the relation

$$\text{ACOSH}(\text{COSH } y) = y$$

will fail badly for y values such that

$$y > \text{INC}(\text{REAL}(\text{LARGE})/2)$$

IV.B.13 ATANH

Valid Input Value Range: -1.0 to 1.0

Output range: -REAL'LARGE to REAL'LARGE

If the argument, x , is in the interval -0.25 to 0.25 a polynomial approximation is used for ATANH. In the interval $0.25 < |x| < 1.0$ the relation $\ln((1+x)/(1-x))/2$ is used.

For the invalid input region $|x| > 1.0$, the value \pm REAL'LARGE is returned and no error halt or exception is raised; this point is discussed in Section IV.A.4.

IV.C. Verification of the Function Package

Short of proving that each function given here is correct, the best that can be done is to check the correctness of the value computed for a large number of arguments for each function. Since at this point we have no way of obtaining check values internally in Ada, it appears that to check them, we must write them out and verify each value by hand or by another computer program in a different language such as FORTRAN.

There is, however, another good method of verifying the correctness of a large number of values of a function and that is by checking certain addition theorems for these functions. For example the sine and cosine should obey the relation $\sin^2 x + \cos^2 x = 1$ for all x . This, however, is not a very good check since the relative error of the smaller of these could be very large but not be observed. A better check are the "triple relations". For example

$$\sin 3x = 3 \sin x - 4 \sin^3 x$$

can be used to verify the sine function. The triple relations have the advantage of requiring just one of the functions at a time and allows it to be

checked against itself. Table IV.C-I gives a set of triple relations for the functions considered here. A very good verification procedure would be to check the triple relation of each function for several thousand (random) points uniformly distributed over a range that would ensure the exercising of all branches of the function code. Let us emphasize again that this has not been done yet due to lack of time and not having a suitable Ada compiler. Only the exponential function has been executed and partially verified; Figures V.B-I and II give triple relation error curves of EXP for a limited sample of arguments.

V. Critique

V.A. What Went Wrong

The most serious difficulty encountered by this project was the lack of an adequate Ada compiler. In fact not until 30 days before the final termination date of the project (60 days after the original termination date, a three month no cost extension was requested and granted), did we obtain the Ada/ED compiler. Ada/ED (Version 11.4) was obtained through NTIS and is considered a preliminary unvalidated version intended for education and experimental use only. And while it was extremely complete, it did have a variety of bugs or errors; those we uncovered are listed in Table V.A-I. These errors, while not serious once they were understood, contributed significantly to our confusion. We were, in fact, learning about Ada and the difficulties of untangling our mistakes and misapprehensions about Ada from errors in Version 11.4 proved a real strain and lead to the consumption of vast amounts of computer time. (Ada/ED) is an experimental version and not designed for production use -- it is very slow -- by a factor of about 10^4 to

Table IV.C-I

The "Triple Relations" for Verifying the Accuracy of the Elementary Functions.

$$\sin 3x = 3 \sin x - 4 \sin^3 x$$

$$\cos 3x = -3 \cos x + 4 \cos^3 x$$

$$\tan 3x = (3 \tan x - \tan^3 x)/(1 - 3 \tan^2 x)$$

$$e^{3x} = (e^x)^3$$

$$\ln 3x = \ln 3 + \ln x \text{ or } \ln x^3 = 3 \ln x$$

$$\sqrt{3x} = \sqrt{3}\sqrt{x} \quad \text{or} \quad \sqrt{x^3} = (\sqrt{x})^3$$

$$\sinh 3x = 3 \sinh x + 4 \sinh^3 x$$

$$\cosh 3x = -3 \cosh x + 4 \cosh^3 x$$

$$\tanh 3x = (3 \tanh x + \tanh^3 x)/(1 + 3 \tanh^2 x)$$

$$3 \sin^{-1} x = \sin^{-1}(3x - 4x^3)$$

$$3 \cos^{-1} x = \cos^{-1}(-3x + 4x^3)$$

$$3 \tan^{-1} x = \tan^{-1}((3x - x^3)/(1 - 3x^2))$$

$$3 \sinh^{-1} x = \sinh^{-1}(3x + 4x^3)$$

$$3 \cosh^{-1} x = \cosh^{-1}(-3x + 4x^3)$$

$$3 \tanh^{-1} x = \tanh^{-1}((3x + x^3)/(1 + 3x^2))$$

Table V.A-I

A list of known flaws (as of 10/26/81) in the Ada/ED Compiler Version 11.4

Compile Time:

1. Generic and actual parameters may not have the same name.
2. LONG_FLOAT not implemented.
3. Won't handle exponentiation of a universal constant.

Run Time:

1. Won't multiply by a floating point number if its value is 0.0.
2. INTEGER truncates instead of rounds (sometimes?).
3. Universal constants as parameters of functions or procedures cause a RUN time error halt, not a compile time error.

10⁵ (in execution) over what will be required in a production mode.)

The net result of not obtaining a compiler until so late in the program and then to have it be somewhat flawed has been that the functions and packages here are not very well debugged. In fact we will have to include the disclaimer that "this package of functions is an unvalidated early version and is intended for experimental use only". We would like to be able to claim at this point that each of the functions has been thoroughly debugged and can be certified correct; but this is not at all the case. The best we can claim is that they have been syntax checked; except for the EXP function, none of the functions given here have been executed. Preliminary debugging of the EXP function has been accomplished and error curves obtained showing that EXP does meet the error tolerances requested; however, even here more checking should be done before it is released for incorporation into a "live" system.

V.B. What Went Right

The main bright note of this project is that most of the ideas about how to use Ada to build library packages seem to have checked out. To summarize these briefly:

1) Packages; the packaging feature of Ada works nicely and is just what is needed for writing library subroutines.

2) Generics; also work well and provide a mechanism for constructing variable accuracy routines where the error tolerance is effectively supplied by the user at the time the package is integrated into his total program. This has been demonstrated explicitly for the EXP function where the error curves show how the accuracy depends on the users specification of the number of digits in his REAL type variables; see Figures V.B-I and II.

3) Overloading; this was a useful and convenient device but was not really crucial to setting up the library routines. The main advantage of overloading was that it allowed mixed mode arithmetic (most of the arithmetic operators were overloaded for various combinations of INTEGER, REAL, FLOAT and LONG_FLOAT).

4) Function and procedures; these constructs also worked well and of course were indispensable in setting up an elementary math function library.

5) Dynamic Arrays; worked well and were especially useful for library routines in allocating exactly the needed storage (for the polynomial coefficients for example).

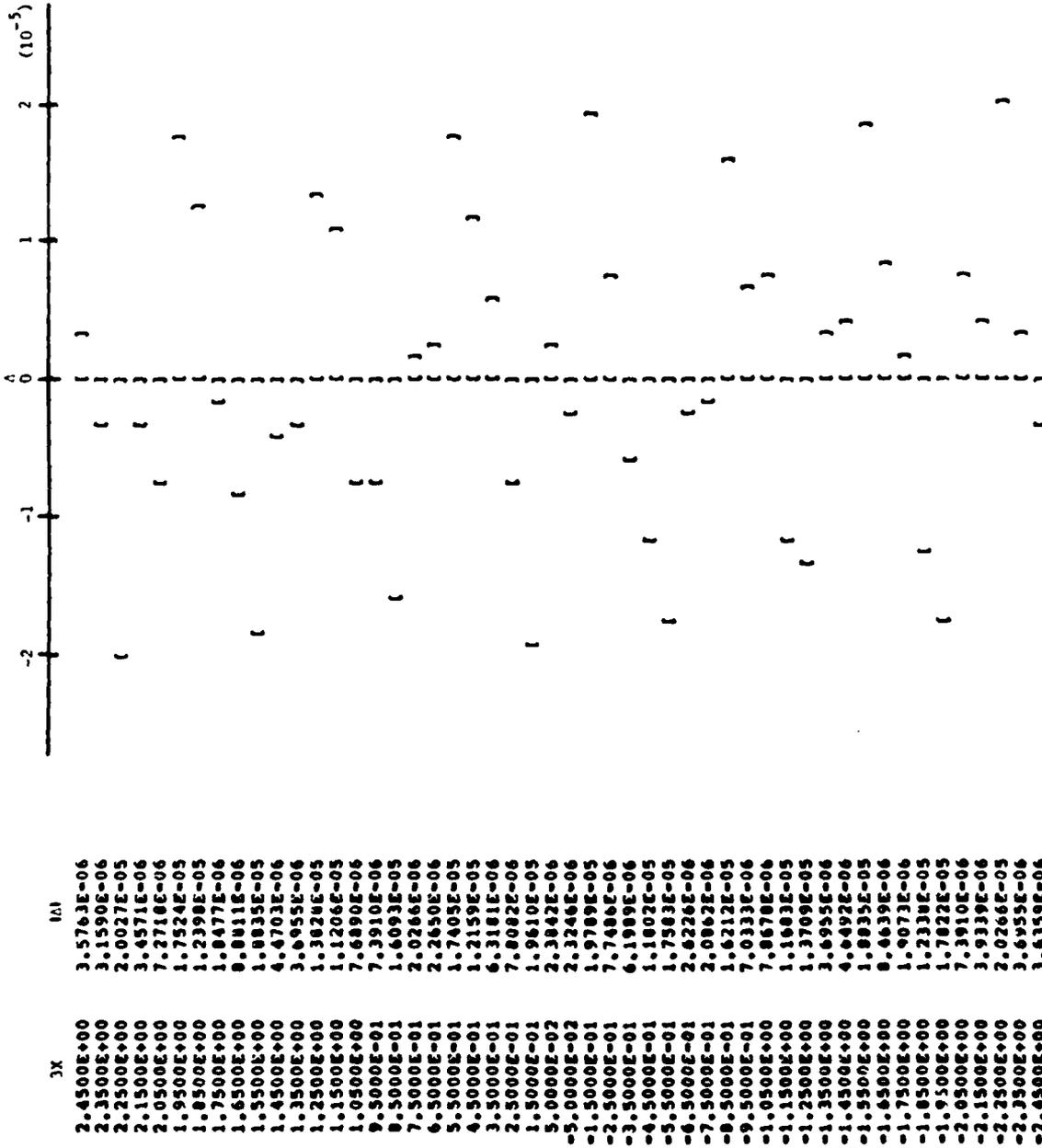
6) Isolation of machine dependent operation; most of the operations expected to be machine dependent were isolated into small, short functions. These are mainly the bit-picking operations and are to be tailored by each installation to their particular hardware. By using the pragma INLINE, these operations can be made run time efficient. However, since the INLINE pragma is not implemented in the Ada/ED version 11.4, it was not possible to demonstrate explicitly here that this works well. We were able to demonstrate that this is not a difficult or unnatural way to proceed.

As mentioned earlier we were able to demonstrate that the elementary function package is syntactically correct and have been able to execute a reduced package containing just the EXP function. Figures V.B-I and II show error curves for the EXP function. (These error curves were generated by comparing e^{3x} with $(e^x)^3$ for 50 values of x . Note that $\Delta = e^{3x}/(e^x)^3 - 1$ can be up to four times the error tolerance since cubing e^x will triple the relative error and the error in e^{3x} can be in the opposite direction.) It would be desirable to run several more such curves at different (higher) accuracy requests before certifying the correctness of the EXP

Figure V.B-I

An error curve for the EXP function.

The measure of the relative error, $\Delta = \text{EXP}(3X)/\text{EXP}(X)**3 - 1$, is plotted here vs X. REAL'DIGITS = 5 here giving an error tolerance of 10^{-5} .



Execution complete
 Execution time: 9.05 seconds
 I-code statements executed: 2295

function; and of course similar tests must be run on the rest of the library functions before they can be used with any confidence.

V.C. Unresolved Problems

There is one major unsolved problem associated with the particular method of constructing function packages as given here. This has to do with memory space and the fact that Ada does not seem to have anything similar to the OVERLAY feature of FORTRAN. In setting up the elementary functions SIN, COS, EXP, etc., a set of Chebyshev constants are introduced for each function; these are used to calculate the coefficients of the power series or polynomial approximation. These new constants need to be retained for the remainder of the execution of the program, but the Chebyshev constants need not be retained; the space allocated to the Chebyshev coefficients could be abandoned or reused for something else, except that Ada has no mechanism for doing this. Also there are program segments that are needed only at initialization time; these too could be abandoned once initialization has been completed and the space reused, but again there is no mechanism for doing this.

In a paging environment on a large computer such as the VAX, this is no problem -- segments that are no longer needed are eventually paged out and never brought back into main memory again. For the anticipated Ada applications however, say for executing an Ada program on a microprocessor with only 4K bytes, space is crucial and it is imperative that no longer needed data or program areas in memory be reusable. We do not know yet how in Ada to reuse either data or program memory area no longer needed, unless the system has some kind of automatic paging arrangement.

11. Results and Conclusions

The principal results of this effort show that the inclusion of library components is well supported by the various Ada constructs such as packages, generics, procedures, functions and records. However, the present state of Ada compilers is such that these facilities of Ada cannot yet be tested. In particular, one would like to know that the machine and accuracy independent techniques used here provide elementary math function facilities that run as efficiently as hand written assembly language routines. Such comparisons cannot be made at this time and will depend heavily on the optimization effectiveness of the particular quality Ada compiler. Assuming that the optimization procedures to be used are reasonably effective, the family of machine and accuracy independent routines described here should be as efficient as hand written assembly language routines.

111 Recommendations for Further Study

There are several issues that need yet to be completed before a final variable λ library of elementary functions can be established for general use in a production environment.

1) The existing routines in the package presented here need to be better thoroughly examined and audited. This was not done here because of the quality of the λ compiler available and because of time limitations.

2) The package here needs to be completed with a production quality λ compiler and the resulting machine code examined and compared to hand written assembly language routines for efficiency. This would be a measure both of the effectiveness of the λ compiler and optimizer and of the elementary function routines themselves.

3) The routines which here are meant for floating point operations, similar routines for integer and fixed point operations need to be written and included in detail for efficiency.

4) The routines presented here are not well organized for running time, although a routine for a λ primitive needs to be examined. A complete analysis of the hardware and software requirements will lead to a different organization of the program.

5) Finally, in an ideal λ environment, the breadth of the function library needs to be considerably expanded and should also include, for example, functions of complex variables and values.

LIST OF REFERENCES

1. DeLone, Irving Allen, "Mathematical Analysis for Computer Solution of Nonlinear Problems," 1957 (GPO: 1957)
2. DeLone, Irving Allen, "Mathematical Approximations for Nonlinear Problems," 1958 (GPO: 1958)
3. DeLone, Irving Allen, "Approximations for Digital Computers," "Proceedings of the Institute of Electrical and Electronics Engineers," 1957 (GPO: 1957)
4. DeLone, Irving Allen, "The Application of the Method of Finite Differences and Other Numerical Methods to Problems in Approximate Analysis," "Mathematics," 1951 (GPO: 1951)
5. DeLone, Irving Allen, "The Approximation of Functions of Several Variables," 1956 (GPO: 1956)
6. DeLone, Irving Allen, "Mathematical Methods for Numerical Approximation," "Mathematics," 1954 (GPO: 1954)

1950

1. The first part of the report is devoted to a general survey of the work done during the year.

The following are the main results of the work done during the year: (1) The first part of the report is devoted to a general survey of the work done during the year. (2) The second part of the report is devoted to a detailed study of the work done during the year. (3) The third part of the report is devoted to a detailed study of the work done during the year.

ADAfile: Fun1.ADA
 AISfile: Fun1.AIS
 LISfile: Fun1.LIS

```

1  -- Disclaimer-- This package of functions is an unvalidated
2  -- early version and is intended for experimental use only. . .
3  --
4  -- Many of the constructs used in this package at least appeared and
5  -- unnatural but were necessary to overcome the flaws in the
6  -- ADA/ED Compiler (version 11.4, the only one available at the
7  -- time these packages were being prepared.
8  --
9  --
10 --
11 --
12 --
13 --
14 --
15 --
16 --
17 --
18 --
19 --
20 --
21 --
22 --
23 --
24 --
25
26 generic type array is array of;
27 type array is array of;
28
29
30 package Fun1 is
31 type array is array of;
32 function array1 (array: array) return array;
33 function array2 (array: array) return array;
34 function array3 (array: array) return array;
35 function array4 (array: array) return array;
36 function array5 (array: array) return array;
37 function array6 (array: array) return array;
38 function array7 (array: array) return array;
39 function array8 (array: array) return array;
40 function array9 (array: array) return array;
41 function array10 (array: array) return array;
42 function array11 (array: array) return array;
43 function array12 (array: array) return array;
44 function array13 (array: array) return array;
45 function array14 (array: array) return array;
46 function array15 (array: array) return array;
47 function array16 (array: array) return array;
48 function array17 (array: array) return array;
49 function array18 (array: array) return array;
50 function array19 (array: array) return array;
51
52

```

```

53     function EXP(X:REAL) return REAL;
54     function LN(X:REAL) return REAL;
55     function SQRT(X:REAL) return REAL;
56     function ATAN(X:REAL) return REAL;
57     function SIN(X:REAL) return REAL;
58     function COS(X:REAL) return REAL;
59     function TAN(X:REAL) return REAL;
60     function ASIN(Y:REAL) return REAL;
61     function ACOS(X:REAL) return REAL;
62     function ATANH(X:REAL) return REAL;
63     function SINH(X:REAL) return REAL;
64     function COSH(X:REAL) return REAL;
65     function TANH(X:REAL) return REAL;
66     function ASINH(X:REAL) return REAL;
67     function ACOSH(X:REAL) return REAL;
68
69     end ALL_FUNCTIONS;
70
71
72
73     package BODY_FUNCTIONS is
74
75
76     -- 0 is overloaded for INT and
77     -- 1 is overloaded for R/I
78     -- where I is INTEGER and R is REAL or FLOAT or LONG_FLOAT.
79
80
81     function ***(I:INTEGER; R:REAL) return REAL is --
82     -- pragma INLINE;
83     begin return REAL(I)*R;
84     end ***;
85
86
87     function ***(X:REAL; I:INTEGER) return REAL is --
88     -- pragma INLINE;
89     begin return X**REAL(I);
90     end ***;
91
92
93     function ***(I:INTEGER; R:FLOAT) return FLOAT is --
94     -- pragma INLINE;
95     begin return FLOAT(I)*R;
96     end ***;
97
98
99     function ***(X:FLOAT; I:INTEGER) return FLOAT is --
100    -- pragma INLINE;
101    begin return X**FLOAT(I);
102    end ***;
103
104
105    function ***(I:INTEGER; R:LONG_FLOAT) return LONG_FLOAT is --
106    -- pragma INLINE;
107    begin return LONG_FLOAT(I)*R;
108    end ***;
109

```

```

110
111 function A2LONG_FLOAT(I:INTEGER) return LONG_FLOAT is --
112 -- pragma INLINES;
113 begin return A/DLONG_FLOAT(I);
114 end A2LONG_FLOAT;
115
116
117 function BASE_2_EXPON(A:REAL) return INTEGER is --
118 -- pragma INLINES;
119 -- a function that gives the base 2 exponent
120 -- of A, such that 0.5 <= ABS(X)/2**BASE_2_EXPON(X) < 1.0,
121 -- (provided X is not 0);
122 -- it should be machine dependent and written in assembly code.
123 -- pragma INLINES;
124     Z:REAL := ABS(X);
125     I:INTEGER := 1;
126 begin if A /> 0.5 then
127         while Z >= 1.0
128             loop Z := Z/2;
129                 I := I + 1;
130             end loop;
131         while Z < 0.5
132             loop Z := 2*Z;
133                 I := I - 1;
134             end loop;
135         end if;
136     return I;
137 end BASE_2_EXPON;
138
139
140 TOL:constant INTEGER := 22; -- larger because of bug in NYU compiler versi
141
142 function TOL_POWER_OF_2(I:INTEGER) return INTEGER is --
143 -- pragma INLINES;
144 -- a function equivalent to 1<sup>I</sup>;
145 -- it could be inlined in assembly code.
146 begin return 2**I;
147 end TOL_POWER_OF_2;
148
149
150 function TOL_POWER_OF_2(I:INTEGER) return REAL is --
151 -- pragma INLINES;
152 -- a function equivalent to 2<sup>I</sup>;
153 -- it could be inlined in assembly code.
154 begin return REAL(TOL_POWER_OF_2(I));
155 end TOL_POWER_OF_2;
156
157
158 function CO-PIER_REAL(I:REAL) return REAL is --
159 -- pragma INLINES;
160 -- a function equivalent to X/TOL_POWER_OF_2(BASE_2_EXPON(X));
161 -- it could be inlined in assembly code.
162 begin return X/REAL(TOL_POWER_OF_2(BASE_2_EXPON(X)));
163 end CO-PIER_REAL;
164
165
166 -----

```

```

167 --
168 -- OPTIMIZES;
169 -- some of these could best be written in assembly code.
170 --
171 -----
172
173
174 function COND(
175     B:BOOLEAN;
176     X,Y:REAL)
177     return REAL is --
178     -- pragma INLINE;
179     begin  if B then return X;
180            else return Y;
181            end if;
182     end COND;
183
184
185 function COND(
186     B:BOOLEAN;
187     X,I:INTEGER)
188     return INTEGER is --
189     -- pragma INLINE;
190     begin  if B then return X;
191            else return Y;
192            end if;
193     end COND;
194
195
196 function ODD(I:INTEGER) return BOOLEAN is --
197     -- pragma INLINE;
198     -- Better in assembly code.
199     begin  return I mod 2 /= 0;
200     end ODD;
201
202
203 function ROUND(X:REAL) return INTEGER is --
204     -- pragma INLINE;
205     -- NOTE: Version 11.4 of compiler TRUNCATES TOWARDS ZERO
206     -- instead of rounding for INTEGER.
207     begin  if X > 0.0 then return INTEGER(X + 0.5);
208            else return INTEGER(X - 0.5);
209            end if;
210     end ROUND;
211
212
213 function "rem"(A, B:REAL) return REAL is --
214     -- pragma INLINE;
215     -- Better in assembly if there is a machine
216     -- code instruction for remainder divide of FLOATs.
217     begin  return(A - B*REAL(ROUND(A/B)));
218     end "rem";
219
220
221 function SIGN(Z, X:REAL) return REAL is --
222     -- pragma INLINE;
223     begin  if Z < 0.0 then return -X;

```

```

224         elsif Z > 0.0 then return X;
225         else return 0.0;
226         end if;
227     end SIGN;
228
229
230     function POLY(X:REAL; A:A_REAL) return REAL is --
231     -- pragma INLINE; -- ?
232         TEMP:REAL := A(A^LAST);
233     begin for J in reverse 0..A^LAST-1
234         loop TEMP := A(J) + X*TEMP;
235         end loop;
236         return TEMP;
237     end POLY;
238
239
240     function ODD_POLY(X:REAL; A:A_REAL) return REAL is --
241     -- pragma INLINE;
242     begin return X*POLY(X*X, A);
243     end ODD_POLY;
244
245
246     function EVEN_POLY(X:REAL; A:A_REAL) return REAL is --
247     -- pragma INLINE;
248     begin return POLY(X*X, A);
249     end EVEN_POLY;
250
251
252
253     -----
254     --
255     --           The Elementary Functions in ADA
256     --
257     --           In the past the elementary functions have been written in
258     --           assembly code and incorporated into the higher order language
259     --           program by being intrinsics of the language and were handled as
260     --           part of the compiler. The accuracy was fixed at the accuracy of
261     --           the machine for which the compilation was done. In Ada, however,
262     --           neither the machine nor the accuracy is known until compile time.
263     --           In most cases these functions are mapped onto some reduced range
264     --           and then approximated over that range by a Chebyshev polynomial
265     --           derived power series. (The advantage of the Chebyshev polynomials
266     --           are that they give a smaller maximum error over the approximation
267     --           range than any other polynomial of the same degree.)
268     --           The Chebyshev polynomials themselves are not used but the sums
269     --           of their coefficients of like powers of the variable are used as
270     --           the constants in the power series. Since the Chebyshev polynomials
271     --           contain all powers up to the order of the polynomial, adding one
272     --           more term (to get more accuracy for example) changes all the terms
273     --           in the power series. This means that the number and values of the
274     --           coefficients must be calculated at compile time (or later) when
275     --           required accuracy is known.
276     --
277     --
278     -----
279
280

```

```

281
282 subtype INT_1_2 is INTEGER range 1..2;
283 subtype INT_0_1 is INTEGER range 0..1;
284 KMAX:constant := 10;
285 type A_LONG_FLOAT is array(INTEGER range <>) of LONG_FLOAT;
286
287
288 PI      :constant := 3.14159_26535_89793_23846_26433;
289 SORT_2  :constant := 1.41421_35623_73095_04880_16887;
290 LN_2    :constant := 0.69314_71805_59945_30941_72321;
291
292
293 -----
294 --
295 --      CHEBYSHEV EXPANSION
296 --
297 --
298 -- Let
299 --      f(x) = Sum(H(k)*T(k,x/n)),      -d<=x<=d
300 --              (0<=k<=n)
301 --
302 -- where
303 --      T(k,x/n) =      Sum(C(k,j)*(x/n)**j)
304 --                      (0<=j<=k)
305 --
306 --      h(k) = (2/n1)*Integral(f(x)*t(k,x/d)/sqrt(1-(x/d)**2)), k>0
307 --                      (-1<=x/d<=1)
308 --
309 --      = (2/r1)*Integral(f(a*cos(y))*cos(k*y))
310 --                      (0<=y<=pi)
311 --
312 -- then
313 --      t(x) = Sum(      Sum(H(k)*C( ,j)*(x/d)**j))
314 --              (0<=k<=r) (0<=j<=k)
315 --
316 --      = Sum(A(j)*x**j)
317 --              (0<=j<=n)
318 --
319 -- where
320 --      A(j) = (1/d)**j*Sum(H(k)*C(k,j))
321 --              (j<=k<=n)
322 --
323 --
324 -----
325
326 package REAL_10 is new FLOAT_10(REAL);
327 use REAL_10;
328
329
330 procedure ICH_SUM(
331     A:out A_REAL;
332     H:A_LONG_FLOAT;
333     L:INT_1_2;
334     I:(INT_0_1)) is --
335     --  L = 2,  I = 0  gives even power series,
336     --  L = 2,  I = 1  gives odd power series,
337     --  L = 1,  I = 0  gives both even and odd terms,

```

```

338      -- L = 1, I = 1 is undefined.
339      -- The CHEBYSHEV constants:
340      function C(N, J:INTEGER) return INTEGER is --
341          -- This is not efficient, time wise, but no matter;
342          -- only used to initialize package constants,
343          -- never used in real time.
344          -- This IS efficient space wise, since it eliminates
345          -- the need of storing the sparse matrix C(N, J).
346      begin if N < 0 or else J < 0 or else J > N
347          or else JDD(N+J) then return 0;
348          elsif (J=0 and N=0) or else (N=1 and J=1) then return 1;
349          else return 2*C(N-1,J-1) - C(N-2, J);
350          end if;
351      end C;
352      TEMP:LONG_FLOAT;
353      CTEMP:INTEGER;
354  begin  -- TCH_SUM
355      for J in 0..A^LAST
356      loop  TEMP := 0.0;
357          for K in J..A^LAST
358          loop  CTEMP := C(L*K+1, L*J+1);
359              -- Version 11.4 wont multiply by floating zero.
360              if CTEMP /= 0 then
361                  TEMP := TEMP + CTEMP*b(K);
362              end if;
363          end loop;
364          A(J) := REAL(TEMP*b(-1)**(-L*J-1));
365          PUT(A(J), WIDTH => 15);
366      end loop;
367      new_line;
368  end TCH_SUM;
369
370
371  function LIM(B:A_LONG_FLOAT; U:REAL := 10.0) return INTEGER is --
372      K:INTEGER := 1;
373      REAL_DIGITS:INTEGER := INTEGER(REAL^DIGITS);
374  begin
375      PUT(J, WIDTH => 15); new_line;
376      PUT(2*U, WIDTH => 15); new_line;
377      loop if ABS(REAL(B(K))) < U**(-REAL_DIGITS) then
378          return K - 1;
379          elsif K = B^LAST then return K;
380          end if;
381          K := K + 1;
382      end loop;
383  end LIM;
384
385
386  -- Note B_xxx(-1) = magnitude of range of variable in function xxx;
387  -- 1e
388  --      B_SIN(-1)           = pi/4           B_COS(-1)           = pi/4
389  --      B_ASIN(-1)          = 0.375          B_TAN(-1)           = pi/8
390  --      B_SQRT(-1)          = 0.25           B_EXP(-1)           = 0.25
391  --      B_ATAN(-1)          = sqrt(2) - 1
392  --      B_LN(-1)            = (2 - sqrt(2))/(2 + sqrt(2))
393  --      B_SINH(-1)          = 1.0            B_COSH(-1)          = 1.0
394  --      B_TANH(-1)          = 0.5            B_ATANH(-1)         = 0.25

```

```

395  --      D_ASINH(-1)      = 0.375
396
397
398  -- These constants are believed to be accurate to the 24th decimal digit.
399
400  D_SIN:constant A_LONG_FLOAT(-1.,KMAX) := ( -- Initialize:
401  PI/4,      0.94977_04415_68744_77636_82691,
402  -0.04984_04113_37036_66401_49298,    0.00038_77134_36152_82730_90288,
403  -0.00000_14305_80091_93208_96335,    0.00000_00030_73651_15544_85672,
404  -0.00000_00000_04316_36597_42291,    0.00000_00000_00004_27564_99507,
405  -0.00000_00000_00000_00314_36072,    0.00000_00000_00000_00000_17840,
406  -0.00000_00000_00000_00000_00008,    0.0);
407
408  D_COS:constant A_LONG_FLOAT(-1.,KMAX) := ( -- Initialize:
409  PI/4,      0.85163_19137_04808_01270_04060,
410  -0.14643_06443_90436_88332_07964,    0.00142_14493_11814_64679_69079,
411  -0.00000_49649_68489_82930_00687,    0.00000_00275_76595_60718_73952,
412  -0.00000_00000_47399_49808_16484,    0.00000_00000_00055_49548_54157,
413  -0.00000_00000_00000_04709_70491,    0.00000_00000_00000_00003_02990,
414  -0.00000_00000_00000_00000_00153,    0.0);
415
416  D_TAN:constant A_LONG_FLOAT(-1.,KMAX) := ( -- Initialize:
417  PI/4,      1.02895_66055_75433_75318_77491,
418  0.02738_60540_75084_48492_31238,    0.00043_63685_44497_59071_23490,
419  0.00000_70316_73527_10477_68349,    0.00000_01134_24587_27361_47971,
420  0.00000_00018_23961_53558_18653,    0.00000_00000_29523_14912_17757,
421  0.00000_00000_00475_30384_23712,    0.00000_00000_00007_68431_70544,
422  0.00000_00000_00000_17337_22568,    0.00000_00000_00000_00260_00628);
423
424  D_ATA:constant A_LONG_FLOAT(-1.,KMAX) := ( -- Initialize:
425  0.41421_35623_73095_04880_16887,    0.97341_02303_41744_14048_51820,
426  -0.02595_27204_82648_35837_91975,    0.00081_89529_74923_64028_44632,
427  -0.00001_75374_93335_45973_14001,    0.00000_05405_76653_50161_32801,
428  -0.00000_00175_19588_81541_58313,    0.00000_00005_87028_14861_02751,
429  -0.00000_00000_20142_43147_91343,    0.00000_00000_00703_55322_00659,
430  -0.00000_00000_00024_01890_52757,    0.00000_00000_00000_84272_36557);
431
432  D_ASIN:constant A_LONG_FLOAT(-1.,KMAX) := ( -- Initialize:
433  0.375,      1.01231_73145_25931_81726_33231,
434  0.01252_32424_89649_02473_23589,    0.00021_15261_66854_04819_41129,
435  0.00000_37468_83198_71353_38921,    0.00000_01219_66403_35596_17913,
436  0.00000_00033_93471_34890_65333,    0.00000_00000_49537_56346_24429,
437  0.00000_00000_03025_54681_23203,    0.00000_00000_00044_82716_46257,
438  0.00000_00000_00013_03226_28731,    0.00000_00000_00000_09583_79223);
439
440  D_SINH:constant A_LONG_FLOAT(-1.,KMAX) := ( -- Initialize:
441  1.0,      1.08152_10970_23595_01513_79419,
442  0.04759_32219_22760_47715_49002,    0.00107_94711_71587_13275_02627,
443  0.00000_03748_09280_75475_04916,    0.00000_00720_23614_04923_05362,
444  0.00000_00000_49879_30140_41585,    0.00000_00000_08971_73653_55812,
445  0.00000_00000_02200_09473_15871,    0.00000_00000_00000_00000_68449,
446  0.00000_00000_00000_00000_00035,    0.0);
447
448  D_COSH:constant A_LONG_FLOAT(-1.,KMAX) := ( -- Initialize:
449  1.0,      1.26806_54777_52998_43884_82416,
450  0.06517_32404_32843_13285_02743,    0.00001_49773_22954_29514_68517,

```

```

452 0.00000_00000_01092_12180_06727_95726, 0.00000_00005_50589_60796_73746,
453 0.00000_00000_01037_15223_06775, 0.00000_00000_00001_42375_80163,
454 0.00000_00000_00000_00138_01801, 0.00000_00000_00000_00000_12074,
455 0.00000_00000_00000_00000_00000);
456
457 B_AIA:constant A_LONG_FLTAT(-1.,NMAX) := ( -- Initialize:
458 0.25, 0.98121_62263_72739_36911_57321,
459 -0.03746_11413_53111_75513_26219, 0.00000_04074_71149_92802_75877,
460 0.00000_16447_00571_08841_59411, 0.00000_05231_11543_14572_47830,
461 -0.00000_01126_14754_17044_56745, 0.00000_00000_04401_07642_67180,
462 -0.00000_00000_07343_05191_12464, 0.00000_00000_00177_13606_90129,
463 -0.00000_00000_00004_27331_50434, 0.00000_00000_00000_10307_84646);
464
465 B_AIA:constant A_LONG_FLTAT(-1.,NMAX) := ( -- Initialize:
466 0.25, 1.01072_10205_64314_61394_26297,
467 0.01042_44195_14700_00423_64941, 0.00010_45480_46459_30385_36636,
468 0.00000_12040_22043_43634_59001, 0.00000_00150_98397_67960_61666,
469 0.00000_00001_94208_56856_03688, 0.00000_00000_02718_54483_61932,
470 0.00000_00000_00034_00154_51119, 0.00000_00000_00000_54085_29074,
471 0.00000_00000_00000_00780_59602, 0.00000_00000_00000_00011_39265);
472
473 B_AIA:constant A_LONG_FLTAT(-1.,NMAX) := ( -- Initialize:
474 0.375, 0.44440_16100_57773_90873_57081,
475 -0.01103_04268_01204_03044_08235, 0.00016_44563_41614_66395_75926,
476 -0.00000_32323_14655_32202_31002, 0.00000_00725_29159_60438_06842,
477 -0.00000_00017_59474_72440_49642, 0.00000_00000_44935_08066_33908,
478 -0.00000_00000_01190_32907_91532, 0.00000_00000_00032_40408_37609,
479 -0.00000_00000_00106_90049_55232, 0.00000_00000_00000_02547_89448);
480
481 B_EXP:constant A_LONG_FLTAT(-1.,NMAX) := ( -- Initialize:
482 0.25, 1.19815_13547_68553_88642_78879,
483 0.20644_43645_34926_24734_13946, 0.00494_98316_92862_14978_54488,
484 0.00025_43197_44594_70042_23755, 0.00000_55933_25892_90406_60652,
485 0.00000_00469_00708_10794_31033, 0.00000_00013_94051_06341_84259,
486 0.00000_00000_17314_61308_18070, 0.00000_00000_00147_50506_43298,
487 0.00000_00000_00001_80476_89915, 0.00000_00000_00000_01563_77974);
488
489 B_LONG:constant A_LONG_FLTAT(-1.,NMAX) := ( -- Initialize:
490 0.17157_24752_53809_40239_66225, 1.00497_23620_80994_03273_56472,
491 0.00494_46092_97031_09813_96480, 0.00002_23658_03341_91784_63774,
492 0.00000_01192_75452_10274_22078, 0.00000_00006_92753_58723_18746,
493 0.00000_00000_04232_41471_88321, 0.00000_00000_00026_75000_48798,
494 0.00000_00000_00000_17315_16241, 0.00000_00000_00000_00114_11170,
495 0.00000_00000_00000_00100_76281, 0.00000_00000_00000_00000_00515);
496
497 B_SIN:constant A_LONG_FLTAT(-1.,NMAX) := ( -- Initialize:
498 0.25, 0.85984_66001_02237_79135_66373,
499 0.14590_57349_49680_61465_28198, -0.00623_81235_48338_35862_41206,
500 0.00053_38902_15777_14176_08675, -0.00005_71863_30329_11416_87631,
501 0.00000_88629_79719_25193_60288, -0.00000_08828_51603_94746_00220,
502 0.00000_11189_39467_39664_64765, -0.00000_00165_75365_75860_25774,
503 0.00000_00023_69307_19312_85914, -0.00000_00003_45461_45137_37718);
504
505
506 B_SIN_MAX : constant REAL := REAL(M_SIN(-1));
507 B_COS_MAX : constant REAL := REAL(P_COS(-1));
508 B_TAN_MAX : constant REAL := REAL(R_TAN(-1));

```

```

509 M_ATAN_HAN      : CONSTANT REAL 16 REAL(M_ATAN(-1))
510 M_ASIN_HAN      : CONSTANT REAL 16 REAL(M_ASIN(-1))
511 M_EXP_HAN        : CONSTANT REAL 16 REAL(M_EXP(-1))
512 M_LN_HAN         : CONSTANT REAL 16 REAL(M_LN(-1))
513 M_SINH_HAN       : CONSTANT REAL 16 REAL(M_SINH(-1))
514 M_COSH_HAN       : CONSTANT REAL 16 REAL(M_COSH(-1))
515 M_TANH_HAN       : CONSTANT REAL 16 REAL(M_TANH(-1))
516 M_ATAN_HAN       : CONSTANT REAL 16 REAL(M_ATAN(-1))
517 M_ASINH_HAN      : CONSTANT REAL 16 REAL(M_ASINH(-1))
518 M_SIN_HAN        : CONSTANT REAL 16 REAL(M_SIN(-1))
519
520
521 P_SIN             : A_REAL(2..LIM(M_SIN))
522 P_COS             : A_REAL(2..LIM(M_COS))
523 P_TAN             : A_REAL(2..LIM(M_TAN))
524 P_ATAN           : A_REAL(2..LIM(M_ATAN))
525 P_ASIN           : A_REAL(2..LIM(M_ASIN))
526 P_EXP            : A_REAL(2..LIM(M_EXP))
527 P_LN             : A_REAL(2..LIM(M_LN))
528 P_SINH           : A_REAL(2..LIM(M_SINH, 1,10))
529 P_COSH           : A_REAL(2..LIM(M_COSH))
530 P_TANH           : A_REAL(2..LIM(M_TANH))
531 P_ATAN           : A_REAL(2..LIM(M_ATAN))
532 P_ASINH          : A_REAL(2..LIM(M_ASINH))
533
534
535
536 -----
537 --
538 --      Irigonometric transformations
539 --
540 --      sin x = sin(x mod 2pi)
541 --            = sign(x) sin(pi/2 - |x|)
542 --            = sign(x) cos(pi/2 - |x|)
543 --            = sign(x) (cos(|x| - pi/4) + sin(|x| - pi/4))/sqrt(2)
544 --
545 --      cos x = cos(x mod 2pi)
546 --            = -cos(pi/2 - |x|)
547 --            = sign(x/2 - |x|)
548 --            = (cos(|x| - pi/4) - sin(|x| - pi/4))/sqrt(2)
549 --
550 --      tan x = tan(x mod 2pi)
551 --            = sign(x) tan(|x| - pi)
552 --            = sign(x)/tan(pi/2 - |x|)
553 --            = sign(x)(tan(|x| - pi/4) + 1)/(1 - tan(|x| - pi/4))
554 --            = sign(x)(tan(|x|-pi) + tan(pi))/(1 - tan(pi)tan(|x|-pi))
555 --            = 2tan(x/2)/(1 - tan(x/2)**2)
556 --
557 --      asin x = atan(x/sqrt(1 - x**2))
558 --
559 --      acosh x = pi/2 + asin(x)
560 --            = sqrt(2x + x**2) + 3x**3/45 + . . .
561 --      where x = 1 - x
562 --
563 --
564 --      Confluent Series Used
565 --

```

```

566 --      IN X      = (16(2)/2 * 2(1) + F)/(B * F) *
567 --                (1/3)(1 + F)/(1 + F) *
568 --                (1/5)(1 + F)/(1 + F) * . . .
569 --      where F = 1/2 * X(1).
570 --
571 --
572 --      The hyperbolic
573 --
574 --      sinh x = (e^x - e^-x) / 2
575 --
576 --      cosh x = (e^x + e^-x) / 2
577 --
578 --      tanh x = sinh(x) / cosh(x)
579 --
580 --      exp x = e^x
581 --
582 --      atan x = arctan(x)
583 --
584 --      asinh x = sinh^-1(x)
585 --
586 --      acosh x = cosh^-1(x)
587 --
588 --      atanh x = arctanh(x)
589 --
590 --
591 .....
```

```

592
593 --      In addition to defining the function the test, average,
594 --      and error rates are given. The average is based on an assumed
595 --      uniform distribution of either the variable (for exp, ln, arct)
596 --      or the range of the function (inverse trig functions)
597 --      over an appropriate interval.
598 --      The 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000.

```

```

611 UNDEFERRABLE
612 SUPPORTABLE
613
614 .....
```

```

621 function ATN(X) return REAL is
622 -- return PI/2 * X / (1 + X^2)

```


001 001000 000000 000000
002 002000 000000 000000
003 003000 000000 000000
004 004000 000000 000000
005 005000 000000 000000
006 006000 000000 000000
007 007000 000000 000000
008 008000 000000 000000
009 009000 000000 000000
010 010000 000000 000000
011 011000 000000 000000
012 012000 000000 000000
013 013000 000000 000000
014 014000 000000 000000
015 015000 000000 000000
016 016000 000000 000000
017 017000 000000 000000
018 018000 000000 000000
019 019000 000000 000000
020 020000 000000 000000
021 021000 000000 000000
022 022000 000000 000000
023 023000 000000 000000
024 024000 000000 000000
025 025000 000000 000000
026 026000 000000 000000
027 027000 000000 000000
028 028000 000000 000000
029 029000 000000 000000
030 030000 000000 000000
031 031000 000000 000000
032 032000 000000 000000
033 033000 000000 000000
034 034000 000000 000000
035 035000 000000 000000
036 036000 000000 000000
037 037000 000000 000000
038 038000 000000 000000
039 039000 000000 000000
040 040000 000000 000000
041 041000 000000 000000
042 042000 000000 000000
043 043000 000000 000000
044 044000 000000 000000
045 045000 000000 000000
046 046000 000000 000000
047 047000 000000 000000
048 048000 000000 000000
049 049000 000000 000000
050 050000 000000 000000
051 051000 000000 000000
052 052000 000000 000000
053 053000 000000 000000
054 054000 000000 000000
055 055000 000000 000000
056 056000 000000 000000
057 057000 000000 000000
058 058000 000000 000000
059 059000 000000 000000
060 060000 000000 000000
061 061000 000000 000000
062 062000 000000 000000
063 063000 000000 000000
064 064000 000000 000000
065 065000 000000 000000
066 066000 000000 000000
067 067000 000000 000000
068 068000 000000 000000
069 069000 000000 000000
070 070000 000000 000000
071 071000 000000 000000
072 072000 000000 000000
073 073000 000000 000000
074 074000 000000 000000
075 075000 000000 000000
076 076000 000000 000000
077 077000 000000 000000
078 078000 000000 000000
079 079000 000000 000000
080 080000 000000 000000
081 081000 000000 000000
082 082000 000000 000000
083 083000 000000 000000
084 084000 000000 000000
085 085000 000000 000000
086 086000 000000 000000
087 087000 000000 000000
088 088000 000000 000000
089 089000 000000 000000
090 090000 000000 000000
091 091000 000000 000000
092 092000 000000 000000
093 093000 000000 000000
094 094000 000000 000000
095 095000 000000 000000
096 096000 000000 000000
097 097000 000000 000000
098 098000 000000 000000
099 099000 000000 000000
100 100000 000000 000000

```

700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

794  --
795  -----
796
797  function SIN(X:REAL) return REAL is --
798  -- pragma INLINE; -- ?
799      Z:REAL := X;
800  begin  loop --
801          if ABS(Z) <= 3*PI/4 then return SIN_SMALL(Z);
802          elsif ABS(Z) <= PI then Z := SIGN(Z, PI - ABS(Z));
803          else Z := Z rem (2*PI);
804          end if;
805      end loop;
806  end SIN;
807  --      Computation time:
808  --      best:      2n + (A + *)n
809  --      Ave:      0 + 0.5M + (A + *) (n + 1)
810  --      worst:    0 + 1 + (A + *) (n + 1)
811
812  --      Hart, page 117 (range[0, pi/4]):
813  --          n = 3      4      5      6      7
814  --      -log10(err) =  4.5  11.3  14.3  17.5  20.7
815
816  -----
817  --
818  --      COS
819  --
820  --
821  -----
822
823  function COS(X:REAL) return REAL is --
824  -- pragma INLINE;
825  begin  if ABS(X) <= 3*PI/4 then return COS_SMALL(X);
826          end if;
827          return SIN(PI/2 - ABS(X));
828  end COS;
829  --      Computation time:
830  --      best:      4 + (A + *)n
831  --      Ave:      0 + A + 0.5M + (A + *) (n + 1)
832  --      worst:    0 + (A + *) (n + 2)
833
834  --      Hart, page 114 (range[0, pi/2]):
835  --          n = 3      4      5      6      7
836  --      -log10(err) =  7.5  10.3  13.2  16.3  19.5
837
838  -----
839  --
840  --      TAN
841  --
842  --
843  -----
844
845  function TAN(X:REAL) return REAL is --
846  -- pragma INLINE; -- ?
847      Z:REAL := X;
848  begin  loop --
849          if ABS(Z) <= 5*PI/8 then return TAN_SMALL(Z);
850          elsif ABS(Z) <= PI then Z := SIGN(Z, ABS(Z) - PI);

```

```

851      ... := ...;
852      ...;
853      ...;
854      ...;
855      ...;
856      ...;
857      ...;
858      ...;
859
860      ...;
861      ...;
862      ...;
863
864
865
866
867
868
869
870
871  ! ...;
872  -- ...;
873  ...;
874  ...;
875  ...;
876  ...;
877  ...;
878  ...;
879  ...;
880  ...;
881  -- Computation time:
882  -- nest: 2 + (A + M)n
883  -- Ave: 1.75(A + M) + n + (A + M)(n + 1)
884  -- worst: 2 + A + (A + M)(n + 2)
885
886  -- ...;
887  -- ...;
888  -- ...;
889
890  -- ...;
891
892
893
894
895
896
897
898
899  function ASIN(I:REAL) return REAL is --
900  -- pragma INLINE? == ?
901  begin
902    -- assert ABS(Y) <= 1.0;
903    if ABS(Y) < P_ASI*_RAN then return ODD_POLY(Y, P_ASIN);
904    elsif ABS(Y) >= 1.0 then return SIGN(Y, PI/2);
905    end if;
906    return ATAN(Y/SQRT((1.0 + Y)*(1.0 - Y)));
907  end ASIN;
908  -- Computation time:

```

```

908 --      Best:  2A + (A + 4)n
909 --      Ave:   2.0625A + 0.5625n + 1.25n +
910 --            0.75(A + 4)(n[atan] + n[sqrt] + n/3 + 1)
911 --      worst: 3A + 4 + (A + 4)(n[atan] + n[sqrt] + 5)
912
913
914 -----
915 --
916 --      ACOS
917 --
918 -----
919
920 ACOS_CONST:REAL;
921
922 function ACOS(X:REAL) return REAL is --
923 -- pragma INLINE;
924     Y:REAL := 1.0 - X;
925 begin
926     -- assert ABS(X) <= 1.0;
927     if ABS(Y) < ACOS_CONST then return SQRT(Y*(2.0 + Y/3));
928     end if;
929     return PI/2 - ASIN(Y);
930 end ACOS;
931
932 --      Computation time:
933 --      Best:  0 + 2A + (A + 4)(n[sqrt] + 3)
934 --      Ave:   2.0675A + 2.5625n + 1.25n +
935 --            0.75(A + 4)(n[atan] + n[sqrt] + n[asin]/3 + 4)
936 --      worst: 3A + 3A + (A + 4)(n[atan] + n[sqrt] + 5)
937
938 -----
939 --      SINH
940 --
941 -----
942
943 function SINH(X:REAL) return REAL is --
944 -- pragma INLINE; -- ?
945     E:REAL;
946 begin
947     if ABS(X) <= H_SINH_RAN then return ODD_POLY(X, E_SINH);
948     end if;
949     E := EXP(X);
950     if E = 0.0 then return -REAL(REAL^LARGE)/2;
951     end if;
952     return (E - 1.0/E)/2;
953 end SINH;
954
955 --      Hart, page 104 (range[0,0.5]):
956 --      n = 0  1  2  3  4  5  6  7
957 --      -log10(err) = 1.7  4.2  7.0  10.1  13.3  16.7  20.3  23.9
958 --      Range[0, 1.0]:
959 --      =      3.0  5.2  7.7  10.3  13.1  16.0  19.1
960
961 -----
962 --
963 --      COSH
964 --

```

```

965 -----
966
967 function COSH(X:REAL) return REAL is --
968 -- pragma INLINE; -- ?
969     E:REAL;
970 begin   if ABS(X) <= H_COSH_RAN then return EVEN_POLY(X, P_COSH);
971         end if;
972         E := EXP(X);
973         if E = 0.0 then return REAL(LARGE)/2;
974         end if;
975         return (E + 1.0/E)/2;
976 end COSH;
977
978 --      Range(0, 1.0):
979 --      n = 1    2    3    4    5    6    7
980 -- -log10(err) = 2.3  4.3  6.7  9.3 12.0 14.8 17.8
981
982 -----
983
984 --
985 --      TANH
986 --
987 -----
988
989 TANH_CONST:REAL;
990
991 function TANH(X:REAL) return REAL is --
992 -- pragma INLINE; -- ?
993     E2X:REAL;
994 begin   if ABS(X) <= H_TANH_RAN then return ODD_POLY(X, P_TANH);
995         elsif ABS(X) >= TANH_CONST then return SIGN(X, 1.0);
996         end if;
997         E2X := EXP(2*X);
998         return (E2X - 1.0)/(E2X + 1.0);
999 end TANH;
1000
1001 -----
1002
1003 --
1004 --      ATANH
1005 --
1006 -----
1007
1008 function ATANH(X:REAL) return REAL is --
1009 -- pragma INLINE; -- ?
1010 begin   -- assert ABS(X) < 1.0;
1011         if ABS(X) <= H_ATANH_RAN then return ODD_POLY(X, P_ATANH);
1012         elsif ABS(X) >= 1.0 then return SIGN(X, LN_REAL_LARGE/2);
1013         end if;
1014         return LN((1.0 + X)/(1.0 - X))/2;
1015 end ATANH;
1016
1017 --      Range(0, 0.251):
1018 --      n = 1    2    3    4    5    6    7    8
1019 -- -log10(err) = 3.0  5.1  7.2  9.7 11.6 13.4 15.3 17.1
1020
1021

```

```

1022 -----
1023 --
1024 --      ASINH
1025 --
1026 -----
1027
1028 function ASINH(X:REAL) return REAL is --
1029 -- pragma INLINE; -- ?
1030 begin   if ABS(X) <= P_ASINH_RAN then return ODD_POLY(X, P_ASINH);
1031         elsif ABS(X) >= SQRT_REAL_LARGE/2 then
1032             return SIGN(X, LN(ABS(X)) + LN_2);
1033         end if;
1034         return SIGN(X, LN(ABS(X) + SQRT(1.0 + X*X)));
1035 end ASINH;
1036
1037 --      Range[0, 0.375]:
1038 --          n = 2    3    4    5    6    7    8    9    10
1039 -- -log10(err) = 5.5  7.1  8.8  10.3  11.1  13.5  15.0  16.6  18.1
1040
1041 -----
1042 --
1043 --      ACOSH
1044 --
1045 --
1046 -----
1047
1048 ACOSH_CONST:REAL;
1049
1050 function ACOSH(U:REAL) return REAL is --
1051 -- pragma INLINE; -- ?
1052     Y:REAL := U - 1.0;
1053 begin   -- assert U >= 1.0;
1054         if U <= 1.0 then return 0.0;
1055         elsif Y <= ACOSH_CONST then return SQRT(Y*(2.0 - Y/3));
1056         elsif U >= SQRT_REAL_LARGE/2 then return LN(ABS(U)) + LN_2;
1057         end if;
1058         return LN(SQRT(Y*(1.0 + U)) + U);
1059 end ACOSH;
1060
1061
1062 TWO_P_FIVE: constant REAL := 2.5; -- needed by compiler version 11.4.
1063
1064 begin   -- Initialization of package tch_fun.
1065
1066
1067     TCH_SUM(P_SIN,  B_SIN,  2, 0);
1068     TCH_SUM(P_COS,  B_COS,  2, 0);
1069     TCH_SUM(P_TAN,  B_TAN,  2, 0);
1070     TCH_SUM(P_ASIN, B_ASIN, 2, 0);
1071     TCH_SUM(P_ATAN, B_ATAN, 2, 0);
1072     TCH_SUM(P_EXP,  B_EXP,  1, 0);
1073     TCH_SUM(P_SINH, B_SINH, 2, 0);
1074     TCH_SUM(P_COSH, B_COSH, 2, 0);
1075     TCH_SUM(P_TANH, B_TANH, 2, 0);
1076     TCH_SUM(P_ASINH, B_ASINH, 2, 0);
1077     TCH_SUM(P_ATANH, B_ATANH, 2, 0);
1078     TCH_SUM(P_LN,  B_LN,   2, 0);

```

```

1079     FCH_SUM(P_SORT, o_SORT, 1, 0);
1080
1081
1082     for I in 0..LI*(3_LW)
1083     loop   P_LB(I) := 2*P_LB(I);
1084     end loop;
1085
1086     ACOS_CONST := 3.0/TWO_P_FIVE**REAL^DIGITS;
1087     ACOSH_CONST := 4.0/TWO_P_FIVE**REAL^DIGITS;
1088     LAGR_CONST := REAL(REAL^DIGITS)*1.2 + LN_2;  -- 1.2 ~= ln(10)/2
1089
1090     PUT(ACOS_CONST, WIDTH => 15);
1091     PUT(ACOSH_CONST, WIDTH => 15);
1092     PUT(LAGR_CONST, WIDTH => 15);
1093     new_line;
1094
1095     LI_REAL_LARGE := LI(REAL(REAL^LARGE));
1096     SORT_REAL_LARGE := SORT(REAL(REAL^LARGE));
1097
1098     PUT(LI_REAL_LARGE, WIDTH => 15);
1099     PUT(SORT_REAL_LARGE, WIDTH => 15);
1100     new_line;
1101
1102 end FIB_FIB;
1103
1104
1105 -----
1106 --
1107 -- REFERENCES
1108 --
1109 -- Rice, John R., "The Approximation of Functions",
1110 -- Addison-Wesley, 1964. QA221 .R5 v1
1111 --
1112 -- Hastings, Cecil Jr, "Approximations for Digital Computers",
1113 -- Princeton University Press, 1955. QA76 .H37 c.2
1114 --
1115 -- Halston, Anthony, "A First Course in Numerical Analysis",
1116 -- McGraw-Hill, 1965. QA297 .H3
1117 --
1118 -- Shviter, Avery Martin, "Chebyshev Methods in Numerical
1119 -- Approximation", Prentice-Hall, 1966. QA221 .S65
1120 --
1121 -- Klovanski, Alexey Nikolaevitch, "The Application of Continued
1122 -- Fractions and Their Generalizations to Problems in
1123 -- Approximation Theory", P. Goodhoff, 1963. QA221 .K473
1124 --
1125 -- Hart, John E., et al., "Computer Approximations," John
1126 -- Wiley and Sons, 1968. QA297 .C64
1127 --
1128 -- Jones, Irving Allen, "Numerical Analysis for Computer
1129 -- Science," North-Holland, 1974. QA297 .D59
1130 --
1131 -----
1132
1133 with ELL_FIB;
1134
1135 procedure test is

```

```
1136         type REEL is digits 5;
1137         type LONG_FLOAT is digits 7;
1138         package NEW_ELE_FUN is new ELE_FUN(REEL, LONG_FLOAT);
1139         use NEW_ELE_FUN;
1140     begin    -- TEST
1141         PUT(" TEST OK ");
1142         new_line;
1143     end TEST;
1144
```

no parse errors detected
Parsing time: 732 seconds

no semantic errors detected
Translation time: 2236 seconds

APPENDIX B

A Listing of the Programs and Subroutines for Computing the Chebyshev Coefficients for the Various Elementary Math Functions

These programs are in FORTRAN but for documentation purposes,
the FLECS listings are first given. The numbers in the right hand column
of the FORTRAN listings show the line to which it corresponds in the FLECS
listing.

```

00001 C
00002 C      THE PROGRAM TRIGOE CALCULATES THE CHEBYSHEV AND POLYNOMIAL
00003 C      COEFFICIENTS FOR ODD OR EVEN (MOSTLY TRIG) FUNCTIONS.
00004 C      ALL COMPUTATIONS ARE IN DOUBLE PRECISION EXCEPT FOR THE
00005 C      ERROR TOLERANCE WHICH IS IN SINGLE.
00006 C      THE ACTUAL COMPUTATIONS ARE CARRIED OUT IN SUBROUTINE POE
00007 C      THE MAIN PROGRAM ONLY DOES THE INTERACTIVE DIALOG TO
00008 C      DETERMINE WHAT IS TO BE COMPUTED AND HOW ACCURATELY.
00009 C
00010 C      THE RANGE OF THE APPROXIMATION TO THE FUNCTION IS FROM
00011 C      -AP TO AP. IF THE USER SPECIFIES AP = 0 OR LESS THEN
00012 C      A DEFAULT AP = 0 IS USED.
00013 C
00014 C      PROGRAM TRIGOE INPUT-OUTPUT
00015 C      IMPLICIT DOUBLE PRECISION (A-H,F-Z)
00016 C      COMMON N, AN, I
00017 C      EXTERNAL FSINH, CSINH,
00018 C      X      FATANH, CATANH,
00019 C      X      FASINH, CASINH,
00020 C      X      FACSHP, CASHP,
00021 C      X      FTANH, ATANH,
00022 C      X      FASINH, ASINH,
00023 C      X      FTANH, ATAN,
00024 C      X      FCSHP, ACSHP,
00025 C      X      FATANH, ATANH,
00026 C      X      FALNT, ALNT,
00027 C      X      FSHACT, SHACT,
00028 C      X      FSHINH, SHINH,
00029 C      X      FCSHP, CSHP,
00030 C      X      FASINH, ASINH
00031 C      DATA PI/3.1415926535897932384626433832795;
00032 C
00033 C      WHILE (.TRUE.)
00034 C      . PRINT *, 'ODD/EVEN FUNCTION ?'
00035 C      . READ *, I
00036 C      . PRINT 10, I
00037 10 . FORMAT(2X,A0)
00038 C      . PRINT *, 'AP = ?'
00039 C      . READ *, AP
00040 C      . PRINT *, AP
00041 C      . CONTINUE
00042 C      . CALL EO1 (I,HSINH)
00043 C      . IF (AP .NE. 0) GO TO 200
00044 C      . CALL EO2 (I,FATANH,ASINH)
00045 C      . GO TO 300
00046 C      . CALL EO3 (I,FATANH)
00047 C      . IF (I .EQ. 2) GO TO 200
00048 C      . CALL EO4 (I,FASINH,ASINH)
00049 C      . GO TO 300
00050 C      . CALL EO5 (I,FTANH)
00051 C      . IF (I .EQ. 3) GO TO 200
00052 C      . CALL EO6 (I,FTANH,ASINH)
00053 C      . GO TO 300
00054 C      . CALL EO7 (I,CSHP)
00055 C      . IF (AP .NE. 0) GO TO 200
00056 C      . CALL EO8 (I,FACSHP,CSHP)
00057 C      . GO TO 300

```

Copy of this document is
 Permitted fully legible reproduction


```

00114 C
00115 C AS IS THE ARRAY OF COEFFICIENTS FOR THE POLYNOMIAL
00116 C APPROXIMATION OF SSSS, COMPUTED IN ACCOF.
00117 C
00118 C THE OLD FUNCTIONS (SIN, TAN, ASIN, ATAN) ARE CONVERTED
00119 C TO EVEN FUNCTIONS (SIN2, TAN2, ASIN2, ATAN2) AND
00120 C ATAN2(X) IN ORDER THAT THE RELATIVE ERROR BE UNIFORMLY
00121 C DISTRIBUTED BY THE UNIFORM APPROXIMATION.
00122 C
00123 C THE APPROXIMATED POLYNOMIAL APPROXIMATION IS VERIFIED
00124 C BY GENERATING AN ERROR MESSAGE OVER THE INTERVAL 0 TO AC.
00125 C
00126 SUBROUTINE POLY (N, X, Y, Z)
00127 IMPLICIT DOUBLE PRECISION (A-H, O-Z)
00128 COMMON /POLY/
00129 DIMENSION I(2), J(2), K(2), L(2)
00130 I(1) = 1
00131 I(2) = 2
00132 J(1) = 1
00133 J(2) = 2
00134 K(1) = 1
00135 K(2) = 2
00136 L(1) = 1
00137 L(2) = 2
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252
00253
00254
00255
00256
00257
00258
00259
00260
00261
00262
00263
00264
00265
00266
00267
00268
00269
00270
00271
00272
00273
00274
00275
00276
00277
00278
00279
00280
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500

```


1. ...
 2. ...
 3. ...
 4. ...
 5. ...
 6. ...
 7. ...
 8. ...
 9. ...
 10. ...
 11. ...
 12. ...
 13. ...
 14. ...
 15. ...
 16. ...
 17. ...
 18. ...
 19. ...
 20. ...
 21. ...
 22. ...
 23. ...
 24. ...
 25. ...
 26. ...
 27. ...
 28. ...
 29. ...
 30. ...
 31. ...
 32. ...
 33. ...
 34. ...
 35. ...
 36. ...
 37. ...
 38. ...
 39. ...
 40. ...
 41. ...
 42. ...
 43. ...
 44. ...
 45. ...
 46. ...
 47. ...
 48. ...
 49. ...
 50. ...
 51. ...
 52. ...
 53. ...
 54. ...
 55. ...
 56. ...
 57. ...
 58. ...
 59. ...
 60. ...
 61. ...
 62. ...
 63. ...
 64. ...
 65. ...
 66. ...
 67. ...
 68. ...
 69. ...
 70. ...
 71. ...
 72. ...
 73. ...
 74. ...
 75. ...
 76. ...
 77. ...
 78. ...
 79. ...
 80. ...
 81. ...
 82. ...
 83. ...
 84. ...
 85. ...
 86. ...
 87. ...
 88. ...
 89. ...
 90. ...
 91. ...
 92. ...
 93. ...
 94. ...
 95. ...
 96. ...
 97. ...
 98. ...
 99. ...
 100. ...

104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200

1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025

1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025

CALL ACDEF:AB, C5555, IC, AN, N/	00182
V = N /	00183
LD 99994 D-1,N	00184
AP: 77 AS12*0	00184
**** CONTINUE	00184
DES DA5501 = AB:01	00186
PRINT * * N * DES *	00187
PRINT * * AN * * AN	00188
DE = AMM:10ES, DE:	00189
MA = 5	00191
DE:1AC = AN 249, *	00192
MA = 2	00193
**** PRINT * * N * * * GO TO 99992	00194
MA = 1	00195
DE:1AC = 1	00196
DE:1AC = 1, MA:	00197
MA = 1, DE:1AC:	00198
DE:1AC = 1, DE:1AC:10000H	00200
**** GO TO 99989	00200
DE:1AC = 1, DE:1AC:10000H	00201
MA = 1, DE:1AC:	00201
DE:1AC = 1, DE:1AC:10000H	00201
**** GO TO 99988	00201
DE:1AC = 1, DE:1AC:10000H	00202
MA = 1, DE:1AC:	00202
DE:1AC = 1, DE:1AC:10000H	00202
**** GO TO 99987	00202
DE:1AC = 1, DE:1AC:10000H	00202
MA = 1, DE:1AC:	00203
DE:1AC = 1, DE:1AC:10000H	00203
**** GO TO 99986	00203
DE:1AC = 1, DE:1AC:10000H	00205
MA = 1, DE:1AC:	00205
DE:1AC = 1, DE:1AC:10000H	00206
**** GO TO 99985	00206
DE:1AC = 1, DE:1AC:10000H	00207
MA = 1, DE:1AC:	00207
DE:1AC = 1, DE:1AC:10000H	00208
**** GO TO 99984	00208
DE:1AC = 1, DE:1AC:10000H	00209
MA = 1, DE:1AC:	00214
DE:1AC = 1, DE:1AC:10000H	00215
**** GO TO 99983	00215
DE:1AC = 1, DE:1AC:10000H	00216
MA = 1, DE:1AC:	00216
DE:1AC = 1, DE:1AC:10000H	00217
**** GO TO 99982	00217
DE:1AC = 1, DE:1AC:10000H	00218
MA = 1, DE:1AC:	00218
DE:1AC = 1, DE:1AC:10000H	00219
**** GO TO 99981	00219
DE:1AC = 1, DE:1AC:10000H	00220
MA = 1, DE:1AC:	00220
DE:1AC = 1, DE:1AC:10000H	00221
**** GO TO 99980	00221

COPY,OSUBOTH

```
-----
00001 C
00002
00003 C      PROGRAM BOTH CALCULATES CHEBYSHEV AND POLYNOMIAL APPROXIMATIONS
00004 C      THAT USE BOTH ODD AND EVEN TERMS IN THEIR APPROXIMATIONS OF A
00005 C      FUNCTION (EXP, LOG, SORT).  ALL COMPUTATIONS ARE IN DOUBLE
00006 C      PRECISION EXCEPT FOR THE ERROR TOLERANCE WHICH IS IN SINGLE.
00007 C      THE ACTUAL COMPUTATIONS ARE CARRIED OUT IN THE SUBROUTINE PBOTH,
00008 C      THE MAIN PROGRAM ONLY DOES THE INTERACTIVE DIALOG TO DETERMINE
00009 C      WHAT IS TO BE COMPUTED AND HOW ACCURATELY.
00010 C
00011 C      THE RANGE OF THE APPROXIMATION TO THE FUNCTION IS FROM AL TO AN,
00012 C      THERE ARE NO DEFAULT VALUES FOR AL OF AN.
00013 C
00014 C      PROGRAM BOTH INPUT OUTPUT
00015 C      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
00016 C      COMMON N, AN, I, AL
00017 C      COMMON/CONSTS/PI, ALN2
00018 C      EXTERNAL FLOGT, ALOG,
00019 C      X      FCBRTT, CBRT,
00020 C      X      FSRACT, SRAC,
00021 C      X      FFWDT, FWDT,
00022 C      X      FEEXP, EXP,
00023 C      X      FE1XT, E1T,
00024 C      X      FASNT, ASIN,
00025 C      X      FSRACT, SRAC,
00026 C      X      FSORT, SORT
00027 C      PI = 3.14159 26535 89793 23646 26433 8327950
00028 C      ALN2 = DLOG(2.000)
00029 C
00030 C      WHILE (.TRUE.)
00031 C      . PRINT *, ' FUNCTION = '
00032 C      . READ *, I
00033 C      . PRINT (O, I)
00034 C      . FORMAT (X,A10)
00035 C      . PRINT *, ' AN = '
00036 C      . READ *, AN
00037 C      . PRINT *, AN
00038 C      . PRINT *, ' AL = '
00039 C      . READ *, AL
00040 C      . PRINT *, AL
00041 C      . CONDITIONAL
00042 C      . . (I) EQ. 10E00T      . CALL FBOOTH(FCBRTT, CBRT)
00043 C      . . (I) EQ. 10E00W      . CALL FBOOTH(FWDT, FWDT)
00044 C      . . (I) EQ. 10E00X      . CALL FBOOTH(FE1T, E1T)
00045 C      . . (I) EQ. 10E00S      . CALL FBOOTH(FASNT, ASIN)
00046 C      . . (I) EQ. 10E00A      . CALL FBOOTH(FASNT, ASIN)
00047 C      . . (I) EQ. 10E00P      . CALL FBOOTH(FEXP, EXP)
00048 C      . . (I) EQ. 10E00R      . CALL FBOOTH(FSRACT, SRAC)
00049 C      . . (I) EQ. 10E00RA      . CALL FBOOTH(FSRACT, SRAC)
00050 C      . . OTHERWISE) PRINT *, ' I'll ASN AGAIN.'
00051 C      . ...FIN
00052 C      ...FIN
00053 C      END
```

(TECS VERSION 02.51)

```

00054 C
00055 C
00056 C
00057 C
00058 C
00059 C
00060 C
00061 C
00062 C
00063 C
00064 C
00065 C
00066 C
00067 C
00068 C
00069 C
00070 C
00071 C
00072 C
00073 C
00074 C
00075 C
00076 C
00077 C
00078 C
00079 C
00080 C
00081 C
00082 C
00083 C
00084 C
00085 C
00086 C
00087 C
00088 C
00089 C
00090 C
00091 C
00092 C
00093 C
00094 C
00095 C
00096 C
00097 C
00098 C
00099 C
00100 C
00101 C
00102 C
00103 C
00104 C
00105 C
00106 C
00107 C
00108 C
00109 C
00110 C
00111 S
00112 C
00113 C

```

SUBROUTINE FROTH COMPUTES THE CHEBYSHEV AND POLYNOMIAL
COEFFICIENTS FOR THE REQUESTED FUNCTION.

SSSS IS THE FUNCTION TO BE APPROXIMATED (IE EXP, SQRT, ETC)
AND FSSST IS $SSSS((AF+AL)*(AF-AL)*COS(THETA))/2)*COS(N*THETA)$
TO BE INTEGRATED OVER THETA FROM 0 TO PI TO GIVE THE
CHEBYSHEV COEFFICIENTS. (AF-AL) AND N ARE AVAILABLE THRU
COMMON.

IC IS THE MATRIX OF COEFFICIENTS FOR CONVERTING CHEBYSHEV
COEFFICIENTS TO POLYNOMIAL COEFFICIENTS; THESE ARE CALCULATED
IN ICH.

CSSSS IS THE ARRAY OF CHEBYSHEV COEFFICIENTS FOR FUNCTION SSSS.

AS IS THE ARRAY OF COEFFICIENTS FOR THE POLYNOMIAL APPROXIMATION
OF SSSS, COMPUTED IN ACCOEF.

THE POLYNOMIAL APPROXIMATION IS VERIFIED BY GENERATING THE
RELATIVE ERROR CURVE OVER THE INTERVAL AL TO AK.

SUBROUTINE FROTH(FSSST, SSSS)
(MPLICIT DOUBLE PRECISION(A-H,I-Z))
COMMON N, AK, I, AL
DIMENSION IC(25, 25), ICNEG(51)
EQUIVALENCE (ICNEG(51), IC(1, 1))
DIMENSION AS(25), ASO(2)
EQUIVALENCE (ASO(2)-AS(1))
DIMENSION CSSSSO(3), CSSSS(25)
EQUIVALENCE (CSSSSO(3), CSSSS(1))
EXTERNAL SSSS, FSSST
INTEGER D
DATA 0/0/, ZERO/0.0D0/
DATA CSSSS/25*0.0D0/
DATA AS/25*0.0D0/
DATA ICNEG/50*1000/, IC/625*1000/
DATA IB/10H /, IS/10H*****//
DATA PI/3.14159 26535 89793 23846 26433 83279D0/
PRINT *, ' QUAD ERR TOL ',
READ *, DE
PRINT *, DE
PRINT *, ' FUNCT ERR TOL ',
READ *, DEF
PRINT *, DEF
N = -1
REPEAT WHILE (DABS(CSSSS(N)).GT.DEF)
. N = N + 1
. CSSSS(N) = 2*RDMBER(ZERO, PI, FSSST, DE)/PI
. IF (N.EQ.0) CSSSS(N) = CSSSS(N)/2
. PRINT 5, N, CSSSS(N)
. FORMAT(15, F35.25)
...FIN

```

00114      N = N - 1
00115      SUM = DABS(CSSSS(0))
00116      DO (J=1,N) SUM = SUM + DABS(CSSSS(J))
00117      PRINT *, N, ' SUMABS(SSSS) = ', SUM
00118      SUM = (CSSSS(0))
00119      DO (J=1,N) SUM = SUM + (CSSSS(J))
00120      PRINT *, N, ' SUM(SSSS) = ', SUM
00121      DEK = DABS(SSSS(AK) - SUM)
00122 C
00123      CALL TCH(IC, N)
00124 C
00125      XBAR = (AK + AL)/2
00126      CALL ACOEF(AS, CSSSS, IC, (AK - AL)/2, N)
00127 C
00128      DES = DABS(SSSS(XBAR) - AS(0))
00129 C
00130      DE = AMAX1(DES, DEK)
00131      PRINT *, N, DES, DEK,
00132      PRINT *, ' AK = ', AK,
00133      PRINT *, ' AL = ', AL
00134 C
00135      JMAX = 5
00136      DELTAX = (AK - AL)/249.9
00137      X = AL
00138      WHILE (X .LT. AK)
00139      .   DEMAXS = 0
00140      .   DEMINS = 0
00141      .   DO (J=1,JMAX)
00142      .   .   X = X + DELTAX
00143      .   .   FSSSS = POLY(X - XBAR, AS, N)
00144      .   .   ESSSS = SSSS(X)
00145 C   .   .   PRINT *, FSSSS, ESSSS
00146      .   .   OERRS = (FSSSS - ESSSS)/ESSSS
00147      .   .   IF (OERRS.GT.0EMAXS) 0EMAXS = OERRS
00148      .   .   IF (OERRS.LT.0EMINS) 0EMINS = OERRS
00149      .   ...FIN
00150 C   .   PRINT *, 0EMINS, 0EMAXS, 0EMINC, 0EMAXC
00151 C   .
00152 C   .   GRAPHICAL OUT PUT OF ERROR CURVE
00153 C   .
00154      .   NS = 30*0EMINS/DE
00155      .   NL = 30*0EMAXS/DE - NS
00156      .   NS = 65 + NS
00157      .   PRINT 10, (IR, J=1,NS), IS, (IB, J=1,NL), IS
00158 10  .   FORMAT(1X, 130A1)
00159      ...FIN
00160      RETURN
00161      END

```

(FLECS VERSION 22.51)

```

-----
00162 C
00163      FUNCTION TWO(X)
00164      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00165      COMMON/CONST5/PI, ALN2
00166      TWO = DEXP(X*ALN2)
00167 C      PRINT*, ' X = ', X, ' TWO(X) = ', TWO
00168      RETURN
00169      END

```

(FLECS VERSION 22.51)

```
00170 C
00171     FUNCTION FTWDT(THETA)
00172     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00173     COMMON N, AK, I, AL
00174     COST = DCOS(THETA)
00175     COSNT = DCOS(N*THETA)
00176     FTWDT = TWO*((AK + AL + (AK - AL)*COST)/2)*COSNT
00177     RETURN
00178     END
```

(FLECS VERSION 22.51)

```
00179 C
00180     FUNCTION FCBRT(THETA)
00181     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00182     COMMON N, AK, I, AL
00183     EXTERNAL CER
00184     COST = DCOS(THETA)
00185     COSNT = DCOS(N*THETA)
00186     FCBRT = CER((AK + AL + (AK - AL)*COST)/2)*COSNT
00187     RETURN
00188     END
```

(FLECS VERSION 22.51)

```
00189 C
00190     FUNCTION CBRT(X)
00191     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00192     CBRT = 0
00193     IF (X .LE. 0) RETURN
00194     CBRT = DSIGN(DERF(DCOS(DCOS(X)/2)), X)
00195     PRINT*, ' X = ', X, ' CBRT(X) = ', CBRT
00196     RETURN
00197     END
```

(FLECS VERSION 22.51)

EOI. 0 FILES. 1 LINES. 893 WORDS.

COPY,COMPIL		
PROGRAM BOTH(INPUT,OUTPUT)		00014
IMPLICIT DOUBLE PRECISION (A-H,F-Z)		00015
COMMON N, AK, I, AL		00016
COMMON/CONSTS/PI, ALN2		00017
EXTERNAL FLOGT, ALOG,		00018
X FCBRTT, CBRT,		00019
X FSOACT, SQAC,		00020
X FTWOT, TWO,		00021
X FEXFT, EXP,		00022
X FEXIXT, EXIX,		00023
X FAASNT, ASIN,		00024
X FSOACT, SQAC,		00025
X FSQRTT, SQRT		00026
PI = 3.14159 26535 89793 23846 26433 83279D0		00027
ALN2 = DLOG(2.0D0)		00028
99999 IF(.NOT.(.TRUE.)) GO TO 99998		00030
PRINT *, ' FUNCTION = ',		00031
READ *, I		00032
PRINT 10, I		00033
10 FORMAT(X,A10)		00034
PRINT *, ' AK = ',		00035
READ *, AK		00036
PRINT *, AK		00037
PRINT *, ' AL = ',		00038
READ *, AL		00039
PRINT *, AL		00040
IF(.NOT.(I .EQ. 10HCBRT)) GO TO 99996		00042
CALL FBOH(FCBRTT, CBRT)		00042
GO TO 99997		00043
99996 IF(.NOT.(I .EQ. 10HTWO)) GO TO 99995		00043
CALL FBOH(FTWOT, TWO)		00043
GO TO 99997		00044
99995 IF(.NOT.(I .EQ. 10HEXIX)) GO TO 99994		00044
CALL FBOH(FEXIXT, EXIX)		00044
GO TO 99997		00045
99994 IF(.NOT.(I .EQ. 10HLOG)) GO TO 99993		00045
CALL FBOH(FLOGT, ALOG)		00045
GO TO 99997		00046
99993 IF(.NOT.(I .EQ. 10HASASIN)) GO TO 99992		00046
CALL FBOH(FAASNT, ASIN)		00046
GO TO 99997		00047
99992 IF(.NOT.(I .EQ. 10HEXP)) GO TO 99991		00047
CALL FBOH(FEXFT, EXP)		00047
GO TO 99997		00048
99991 IF(.NOT.(I .EQ. 10HSQRT)) GO TO 99990		00048
CALL FBOH(FSQRTT, SQRT)		00048
GO TO 99997		00049
99990 IF(.NOT.(I .EQ. 10HSQAC)) GO TO 99989		00049
CALL FBOH(FSOACT, SQAC)		00049
GO TO 99997		00050
99989 PRINT *, ' I'LL ASK AGAIN, '		00050
99997 GO TO 99999		00052
99998 END		00053
SUBROUTINE FBOH(FSSSST, SSSS)		00056
IMPLICIT DOUBLE PRECISION(A-H,F-Z)		00057
COMMON N, AK, I, AL		00058
DIMENSION IC(25, 25), ICNEG(51)		00059
EQUIVALENCE (ICNEG(51), IC(1, 1))		00060
DIMENSION AS(25), ASO(2)		00062
EQUIVALENCE (ASO(2), AS(1))		00063

DIMENSION CSSSS(3), CSSSS(25)	00085
EQUIVALENCE (CSSSS(3),CSSSS(1))	00086
EXTERNAL SSSS, FSSSST	00088
INTEGER O	00089
DAT 0/0/, ZERO/0.000/	00091
DATA CSSSS/25*0.000/	00092
DATA AS/25*0.000/	00094
DATA ICNEG/50*1000/, IC/625*1000/	00094
DATA IB/10H , IS/10H*****/	00095
DATA FI/3.14159 26535 89793 23846 26483 8327900/	00096
PRINT *, * QUAD ERR TOL *	00098
READ *, OE	00099
PRINT *, OF	00100
PRINT *, * FUNCT ERR TOL *	00101
READ *, OEF	00102
PRINT *, OEF	00103
N = -1	00104
GO TO 99998	00105
99999 IF (.NOT. (DABS(CSSSS(N)) .LT. OEF)) GO TO 99997	00106
99998 N = N + 1	00107
CSSSS(N) = 2*NUMBER ZERO* FI + FSSSST* OE/2*F	00108
IF (N.EQ.O) CSSSS(N) = CSSSS(N)/2	00109
PRINT 5, N, CSSSS(N)	00110
5 FORMAT(15, F35.25)	00111
GO TO 99999	00112
99997 N = N - 1	00113
SUM = DABS(CSSSS(O))	00114
DO 99996 I=1,N	00115
SUM = SUM + DABS(CSSSS(I))	00116
99996 CONTINUE	00117
PRINT *, N, * SUMABS(CSSSS) = *, SUM	00118
SUM = (CSSSS(O))	00119
DO 99995 J=1,N	00120
SUM = SUM + (CSSSS(J))	00121
99995 CONTINUE	00122
PRINT *, N, * SUM(CSSSS) = *, SUM	00123
OEN = (DABS(SSSS)*AM) - SUM	00124
CALL TCR(IC, O)	00125
XBAR = (AM + AL)/2	00126
CALL ADEF(AC, CSSSS, IC, (AM + AL)/2, O)	00127
OES = (DABS(CSSSS(XBAR)) - AS(O))	00128
OE = AMAX1(OES, OEN)	00129
PRINT *, N, OES, OEN	00130
PRINT *, * AK = *, AK	00131
PRINT *, * AL = *, AL	00132
JMAX = 5	00133
DELTAX = (AM - AL)/249.*	00134
* AL	00135
99994 IF (.NOT. (X .LT. AM)) GO TO 99993	00136
DEMAXS = 0	00137
DEMINS = 0	00138
DO 99992 J=1,JMAX	00139
X = X + DELTAX	00140
FSSSS = POLY(X - XBAR, AS, N)	00141
ESSSS = SSSS(X)	00142
OERRS = (FSSSS - ESSSS)/ESSSS	00143
IF (OERRS.GT.DEMAXS) DEMAXS = OERRS	00144
IF (OERRS.LT.DEMINS) DEMINS = OERRS	00145
99992 CONTINUE	00146
NS = 30*DEMINS/OE	00147
NL = 30*DEMAXS/OE NS	00148

```

      NS = 55 + NS
      PRINT 19, (IB, J, NS, IS, (IB, J, NS), IS
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197
END

```

FDJ. 0 FILES. 1 RECS. 1413 WORDS.

/COPY,OSUBLIB

```
00001 C
00002 C
00003 C * THIS FUNCTION DOES ROMBERG QUADRATURE ON THE FUNCTION F
00004 C * FROM THE LOWER LIMIT A TO THE UPPER LIMIT B, AND TO THE
00005 C * REQUESTED ACCURACY DE.
00006 C * THIS VERSION DOES NOT USE THE END POINTS.
00007 C
00008 C FUNCTION ROMBRG(A, B, F, DE)
00009 C IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00010 C DIMENSION DD(20)
00011 C INTEGER FOUR(21), 0
00012 C DIMENSION Q(20), RQ(2)
00013 C EQUIVALENCE (RQ(2),Q(1))
00014 C
00015 C DATA D/0/, LMAX/1/, FOUR(1)/4/
00016 C
00017 C Q(0) = 0
00018 C H = (B - A)/2
00019 C IMAX = 1
00020 C
00021 C DO (L=1,20)
00022 C . IF (L.GE.LMAX)
00023 C . . LMAX = L + 1
00024 C . . FOUR(L+1) = 4*FOUR(L)
00025 C . ...FIN
00026 C .
00027 C . SUM = 0
00028 C . DO (I=1,IMAX)
00029 C . . V = (2*I - IMAX - 1.000)/IMAX
00030 C . . U = V*(3 - V*V)/2
00031 C . . UP = 1.500*(1 - V*V)
00032 C . . Y = A + (U + 1)*H
00033 C . . SUM = SUM + F(X)*UP
00034 C . ...FIN
00035 C . SUM = SUM + IMAX*Q(L-1)
00036 C . IMAX = 2*IMAX
00037 C . Q(L) = SUM/IMAX
00038 C .
00039 C . DD(M=1,L)
00040 C . . M = L - 1
00041 C . . Q(K) = (FOUR(M)*Q(K+1) - Q(K))/(FOUR(M) - 1)
00042 C . ...FIN
00043 C .
00044 C . IF (L.GT.1)
00045 C . . ROMBRG = (B - A)*Q(0)
00046 C . .
00047 C . . DO (N=1,L) DD(N) = (Q(0) - Q(N))*(B - A)
00048 C . . PRINT*, ROMBRG, (DD(N), N=1,L)
00049 C . .
00050 C . . DDQ = (ABS(Q(0) - Q(1))*(B - A))
00051 C . . IF (DDQ.LE.DE) RETURN
00052 C . ...FIN
00053 C .
00054 C ...FIN
00055 C RETURN
```

00056 END

(FLECS VERSION 22.51)

```
-----  
00057 C  
00058 C  
00059 C * THIS FUNCTION DOES ROMBERG QUADRATURE ON THE FUNCTION F  
00060 C * FROM THE LOWER LIMIT A TO THE UPPER LIMIT B, AND TO THE  
00061 C * REQUESTED ACCURACY OE.  
00062 C  
00063 C FUNCTION ROMBER(A, B, F, OE)  
00064 C IMPLICIT DOUBLE PRECISION(A-H,P-Z)  
00065 C DIMENSION OD(20)  
00066 C INTEGER FOUR(21), O  
00067 C DIMENSION Q(20), QO(2)  
00068 C EQUIVALENCE (QO(2),Q(1))  
00069 C  
00070 C DATA O/0/, LMAX/1/, FOUR(1)/4/  
00071 C  
00072 C Q(O) = (F(A) + F(B))/2  
00073 C H = B - A  
00074 C IMAX = 1  
00075 C  
00076 C DO (L=1,20)  
00077 C . IF (L.GE.LMAX)  
00078 C . . LMAX = L + 1  
00079 C . . FOUR(L+1) = 4*FOUR(L)  
00080 C . ...FIN  
00081 C .  
00082 C . H = H/2  
00083 C . SUM = 0  
00084 C . DO (I=1,IMAX) SUM = SUM + F(A + H*(2*I - 1))  
00085 C . SUM = SUM + IMAX*Q(L-1)  
00086 C . IMAX = 2*IMAX  
00087 C . Q(L) = SUM/IMAX  
00088 C .  
00089 C . DO (M=1,L)  
00090 C . . K = L - M  
00091 C . . Q(K) = (FOUR(M)*Q(K+1) - Q(K))/(FOUR(M) - 1)  
00092 C . ...FIN  
00093 C .  
00094 C . ROMBER = (B - A)*Q(O)  
00095 C .  
00096 C . DO (K=1,L) OD(K) = (Q(O) - Q(K))*(B - A)  
00097 C . PRINT*, ROMBER, (OD(K), K=1,L)  
00098 C .  
00099 C . ODD = DABS((Q(O) - Q(1))*(B - A))  
00100 C . IF (ODD.LE.OE) RETURN  
00101 C .  
00102 C . ...FIN  
00103 C RETURN  
00104 C END
```

(FLECS VERSION 22.51)

00105 C
00106 C

```
00107     FUNCTION SIN(X)
00108     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00109     SIN = DSIN(X)
00110 C     PRINT*, ' X =', X, ' SIN(X) =', SIN
00111     RETURN
00112     END
```

(FLECS VERSION 22.51)

```
00113 C
00114     FUNCTION TAN(X)
00115     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00116     TAN = DSIN(X)/DCOS(X)
00117 C     PRINT*, ' X =', X, ' TAN(X) =', TAN
00118     RETURN
00119     END
```

(FLECS VERSION 22.51)

```
00120 C
00121     FUNCTION ASIN(X)
00122     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00123     ASIN = DATAN(X/DSQRT(1-X*X))
00124 C     PRINT*, ' X =', X, ' ASIN(X) =', ASIN
00125     RETURN
00126     END
```

(FLECS VERSION 22.51)

```
00127 C
00128     FUNCTION ACOS(X)
00129     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00130     ACOS = DATAN(DSQRT(1-X*X)/X)
00131 C     PRINT*, ' X =', X, ' ACOS(X) =', ACOS
00132     RETURN
00133     END
```

(FLECS VERSION 22.51)

```
00134 C
00135     FUNCTION EXP(X)
00136     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00137     EXP = DEXP(X)
00138 C     PRINT*, ' X =', X, ' EXP(X) =', EXP
00139     RETURN
00140     END
```

(FLECS VERSION 22.51)

```
00141 C
00142     FUNCTION SQRT(X)
```

```
00143      IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00144      SORT = DSORT(X)
00145 C      PRINT*, ' X = ', X, ' SORT(X) = ', SORT
00146      RETURN
00147      END
```

(FLECS VERSION 22.51)

```
-----
00148 C
00149      FUNCTION ATAN(X)
00150      IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00151      ATAN = DATAN(X)
00152 C      PRINT*, ' X = ', X, ' ATAN(X) = ', ATAN
00153      RETURN
00154      END
```

(FLECS VERSION 22.51)

```
-----
00155 C
00156      FUNCTION ALOG(X)
00157      IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00158      ALOG = DLOG(X)
00159      PRINT*, ' X = ', X, ' ALOG(X) = ', ALOG
00160      RETURN
00161      END
```

(FLECS VERSION 22.51)

```
-----
00162 C
00163      FUNCTION COS(X)
00164      IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00165      COS = DCOS(X)
00166 C      PRINT*, ' X = ', X, ' COS(X) = ', COS
00167      RETURN
00168      END
```

(FLECS VERSION 22.51)

```
-----
00169 C
00170      FUNCTION COSH(THETA)
00171      IMPLICIT DOUBLE PRECISION(A-H,F-Z)
00172      COMMON N, AN
00173      COSH = DCOH(THETA)
00174      SINHT = DCOSH(THETA)
00175      ECOSH = DLOSTAN(COSH)*COSHT
00176      RETURN
00177      END
```

(FLECS VERSION 22.51)

00178 C

```

184      FUNCTION FSINT(THETA)
185      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
186      COMMON N, AN
187      LOST = DCOS(THETA)
188      COSNT = ICOS(THETA)
189      FSINT = COSNT
190      IF (LOST.EQ.0) RETURN
191      FSINT = DSIN(LAN(COST)/AN)*COSNT/COST)
192      RETURN
193      END

```

FLECS VERSION 11.511

```

00189 C
00190 C
00191      FUNCTION FROJ(A,N)
00192      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00193      DIMENSION A(1)
00194      FROJ = AN
00195      DO 10 I=1,N-1
00196      FROJ = FROJ*(A(I)+1)
00197      RETURN
00198      END

```

FLECS VERSION 11.511

```

00199 C
00200      FUNCTION FROE(A,N)
00201      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00202      DIMENSION A(1)
00203      N = 4
00204      DO 10 I=1,N
00205      FROE = A(I)
00206      DO 20 J=1,N
00207      FROE = FROE*(A(J)+1)
00208 C
00209      FROE = FROE/A
00210      FROE = FROE/A
00211      FROE = FROE/A
00212      FROE = FROE/A
00213      DO 30 I=1,N
00214      FROE = FROE*(A(I)+1)
00215 C
00216      RETURN
00217      END

```

FLECS VERSION 11.511

```

00218 C
00219      SUBROUTINE FCR(I,N)
00220      DIMENSION I(1),N(1)
00221      INTEGER I
00222      DATA I,N
00223 C
00224      I(1) = 0

```

```

00225      P = D(1, 0) * I
00226      DO 10 J=N
00227         C = P * B(1, J) * (P-2, 0)
00228         D(1, J) = C
00229         P = D(1, J) * I
00230         I = I + 1
00231         IF (MUL(1, J) .EQ. 0) IC(K, J) = 0
00232         ELSE IC(K, J) = (B(1, J) * J - 1) * IC(K-2, J)
00233         PRINT *, J, D(1, J)
00234     END DO
00235     RETURN
00236     END

```

FILE: D:\VERSION 11.01

```

00237
00238
00239      FUNCTION FATANT(THETA)
00240      IMPLICIT DOUBLE PRECISION(A-H,I-Z)
00241      EXTERNAL TAN
00242      COMMON N, AP
00243      COST = DCOS(THETA)
00244      SINT = DSIN(N*THETA)
00245      FATANT = SINT
00246      IF (COST.EQ.0) RETURN
00247      FATANT = (ATAN(AP*COST)/AP)*COSNT/COST
00248      RETURN
00249      END

```

FILE: D:\VERSION 11.01

```

00250
00251
00252      FUNCTION FATANT(THETA)
00253      IMPLICIT DOUBLE PRECISION(A-H,I-Z)
00254      COMMON N, AP
00255      COST = DCOS(THETA)
00256      SINT = DSIN(N*THETA)
00257      FATANT = SINT
00258      IF (COST.EQ.0) RETURN
00259      FATANT = (ATAN(AP*COST)/AP)*COSNT/COST
00260      RETURN
00261      END

```

FILE: D:\VERSION 11.01

```

00262
00263
00264      FUNCTION FASINT(THETA)
00265      IMPLICIT DOUBLE PRECISION(A-H,I-Z)
00266      EXTERNAL ASIN
00267      COMMON N, AP
00268      COST = DCOS(THETA)
00269      COSNT = DCOS(N*THETA)
00270      FASINT = COSNT
00271      IF (COST.EQ.0) RETURN
00272      FASINT = (ASIN(AP*COST)/AP)*COSNT/COST

```

AD-A116 070

GEORGIA INST OF TECH ATLANTA ENGINEERING EXPERIMENT --ETC F/G 9/2
INVESTIGATE CAPABILITY OF ADA HIGHER ORDER PROGRAMMING LANGUAGE--ETC(U)
MAR 82 L J GALLAHER F30602-78-C-0120

UNCLASSIFIED

RADC-TR-82-46

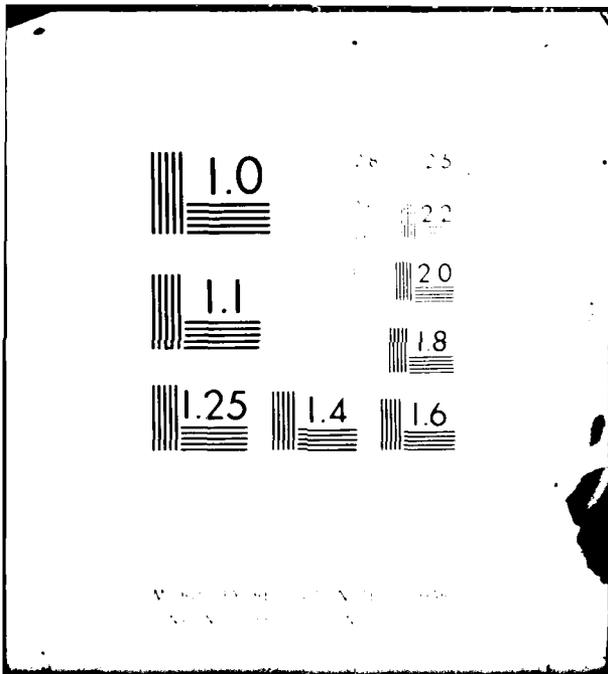
NL

2 OF 2

ADA
116 070



END
DATE
FILMED
07-82
DTIC



VISION TESTING CHART
NATIONAL BUREAU OF STANDARDS

00271 RETURN
00272 END

(FLECS VERSION 22.51)

00273 C
00274 FUNCTION FACOST(THETA)
00275 IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00276 EXTERNAL ACOS
00277 COMMON N, AK
00278 COST = DCOS(THETA)
00279 COSNT = DCOS(N*THETA)
00280 FACOST = ACOS(AK*COST)*COSNT
00281 RETURN
00282 END

(FLECS VERSION 22.51)

00283 C
00284 FUNCTION RASIN(X)
00285 IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00286 EXTERNAL ASIN
00287 RASIN = 1/ASIN(X)
00288 C PRINT*, ' X =', X, ' RASIN(X) =', RASIN
00289 RETURN
00290 END

(FLECS VERSION 22.51)

00291 C
00292 FUNCTION FRASINT(THETA)
00293 IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00294 EXTERNAL RASIN
00295 COMMON N, AK
00296 COST = DCOS(THETA)
00297 COSNT = DCOS(N*THETA)
00298 FRASINT = COSNT
00299 IF (COST.EQ.0) RETURN
00300 FRASINT = COST*RASIN(AK*COST)*AK*COSNT
00301 RETURN
00302 END

(FLECS VERSION 22.51)

00303 C
00304 FUNCTION FALNT(THETA)
00305 IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00306 EXTERNAL ALN
00307 COMMON N, AK
00308 COST = DCOS(THETA)
00309 COSNT = DCOS(N*THETA)
00310 FALNT = COSNT
00311 IF (COST.EQ.0) RETURN

```
00312     FALNT = (ALN(AK*COST)/AK)*(COSNT/COST)
00313     RETURN
00314     END
```

(FLECS VERSION 22.51)

```
00315 C
00316     FUNCTION ALN(Z)
00317     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00318     ALN = DLOG((Z + 1)/(1 - Z))/2
00319 C     PRINT*, ' Z =', Z, ' ALN(Z) =', ALN
00320     RETURN
00321     END
```

(FLECS VERSION 22.51)

```
00322 C
00323     FUNCTION FLOGI(THETA)
00324     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00325     COMMON N, AK, I, AL
00326     COST = DCOS(THETA)
00327     COSNT = DCOS(N*THETA)
00328     FLOGI = DLOG((AK + AL + (AK - AL)*COST)/2)*COSNT
00329     RETURN
00330     END
```

(FLECS VERSION 22.51)

```
00331 C
00332     FUNCTION FSQRTI(THETA)
00333     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00334     COMMON N, AK, I, AL
00335     COST = DCOS(THETA)
00336     COSNT = DCOS(N*THETA)
00337     FSQRTI = DSQRT((AK + AL + (AK - AL)*COST)/2)*COSNT
00338     RETURN
00339     END
```

(FLECS VERSION 22.51)

```
00340 C
00341     FUNCTION FEXP(THETA)
00342     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00343     COMMON N, AK, I, AL
00344     COST = DCOS(THETA)
00345     COSNT = DCOS(N*THETA)
00346     FEXP = DEXP((AK + AL + (AK - AL)*COST)/2)*COSNT
00347     RETURN
00348     END
```

(FLECS VERSION 22.51)

```

00349 C
00350 FUNCTION FAASNT(THETA)
00351 IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00352 EXTERNAL ASIN
00353 COMMON N, AK, I, AL
00354 COST = DCOS(THETA)
00355 COSNT = DCOS(N*THETA)
00356 FAASNT = ASIN((AK + AL + (AK - AL)*COST)/2)*COSNT
00357 RETURN
00358 END

```

(FLECS VERSION 22.51)

```

00359 C
00360 FUNCTION EX1X(X)
00361 IMPLICIT DOUBLE PRECISION (A-H,P-Z)
00362 EX1X = 1
00363 IF (X.NE.0) EX1X = (DEXP(X) - 1)/X
00364 C PRINT*, ' X = ', X, ' EX1X(X) = ', EX1X
00365 RETURN
00366 END

```

(FLECS VERSION 22.51)

```

00367 C
00368 FUNCTION FEX1XT(THETA)
00369 IMPLICIT DOUBLE PRECISION (A-H,P-Z)
00370 COMMON N, AK, I, AL
00371 COST = DCOS(THETA)
00372 COSNT = DCOS(N*THETA)
00373 FEX1XT = EX1X((AK + AL + (AK - AL)*COST)/2)*COSNT
00374 RETURN
00375 END

```

(FLECS VERSION 22.51)

```

00376 C
00377 FUNCTION SQAC(Y)
00378 C NOTE . . THIS FUNCTION COMPUTES 1/2 OF ACOS(1-Y)**2/Y
00379 IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00380 C EXTERNAL ACOS
00381 SQAC = 1
00382 IF (Y.EQ.0) RETURN
00383 C SQAC = ACOS(1 - Y)**2/(2*Y)
00384 SQAC = DATAN(DSQRT(Y*(2 - Y))/(1 - Y)**2/(2*Y)
00385 C PRINT*, 'Y = ', Y, ' SQAC(Y) = ', SQAC
00386 RETURN
00387 END

```

(FLECS VERSION 22.51)

00388 C

```

00389     FUNCTION FSQACT(THETA)
00390     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00391     COMMON N, AK, I, AL
00392     COST = DCOS(THETA)
00393     COSNT = DCOS(N*THETA)
00394     FSQACT = SQAC((AK + AL + (AK - AL)*COST)/2)*COSNT
00395     RETURN
00396     END

```

(FLECS VERSION 22.51)

```

-----
00397 C
00398     FUNCTION SINH(X)
00399     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00400     E = DEXP(X)
00401     SINH = (E - 1/E)/2
00402 C     PRINT*, ' X = ', X, ' SINH(X) = ', SINH
00403     RETURN
00404     END

```

(FLECS VERSION 22.51)

```

-----
00405 C
00406     FUNCTION COSH(X)
00407     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00408     E = DEXP(X)
00409     COSH = (E + 1/E)/2
00410 C     PRINT*, ' X = ', X, ' COSH(X) = ', COSH
00411     RETURN
00412     END

```

(FLECS VERSION 22.51)

```

-----
00413 C
00414     FUNCTION FSINHT(THETA)
00415     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00416     EXTERNAL SINH
00417     COMMON N, AK
00418     COST = DCOS(THETA)
00419     COSNT = DCOS(N*THETA)
00420     FSINHT = COSNT
00421     IF (COST.EQ.0) RETURN
00422     FSINHT = (SINH(AK*COST), AK)*(COSNT/COST)
00423     RETURN
00424     END

```

(FLECS VERSION 22.51)

```

-----
00425 C
00426     FUNCTION FCOSH(THETA)
00427     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00428     EXTERNAL COSH
00429     COMMON N, AP

```

```
00430      COST = DCOS(THETA)
00431      COSNT = DCOS(N*THETA)
00432      FCOSHT = COSH(AK*COST)*COSNT
00433      RETURN
00434      END
```

(FLECS VERSION 22.51)

```
00435 C
00436      FUNCTION ASINH(X)
00437      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00438      EXTERNAL ATANH
00439      ASINH = 0.000
00440      IF (X .EQ. 0) RETURN
00441      ASINH = ATANH(X/DSQRT(1 + X*X))
00442 C      PRINT*, ' X = ', X, ' ASINH(X) = ', ASINH
00443      RETURN
00444      END
```

(FLECS VERSION 22.51)

```
00445 C
00446      FUNCTION FASINH(THETA)
00447      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00448      EXTERNAL ASINH
00449      COMMON N, AK
00450      COST = DCOS(THETA)
00451      COSNT = DCOS(N*THETA)
00452      FASINH = COSNT
00453      IF (COST.EQ.0) RETURN
00454      FASINH = (ASINH+AK*DCOST)/(AK)*(COSNT/COST)
00455      RETURN
00456      END
```

(FLECS VERSION 22.51)

```
00457 C
00458      FUNCTION ACOSH(X)
00459      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00460      EXTERNAL ATANH
00461      ACOSH = ATANH(DSQRT(X*X-1)/X)
00462 C      PRINT*, ' X = ', X, ' ACOSH(X) = ', ACOSH
00463      RETURN
00464      END
```

(FLECS VERSION 22.51)

```
00465 C
00466      FUNCTION FACOSHT(THETA)
00467      IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00468      EXTERNAL ACOSH
00469      COMMON N, AK
00470      COST = DCOS(THETA)
```

```
00471     COSNT = DCOS(N*THETA)
00472     FACOSHT = ACOSH(AK*COST)*COSNT
00473     RETURN
00474     END
```

(FLECS VERSION 22.51)

```
00475 C
00476     FUNCTION ATANH(X)
00477     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00478     ATANH = 0.0D0
00479     IF (X .EQ. 0) RETURN
00480     ATANH = DLOG((1+X)/(1-X))/2
00481 C     PRINT*, ' X =', X, ' ATANH(X) =', ATANH
00482     RETURN
00483     END
```

(FLECS VERSION 22.51)

```
00484 C
00485     FUNCTION FATANH(THETA)
00486     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00487     EXTERNAL ATANH
00488     COMMON N, AK
00489     COST = DCOS(THETA)
00490     COSNT = DCOS(N*THETA)
00491     FATANH = COSNT
00492     IF (COST.EQ.0) RETURN
00493     FATANH = (ATANH(AK*COST)/AK)*(COSNT/COST)
00494     RETURN
00495     END
```

(FLECS VERSION 22.51)

```
00496 C
00497     FUNCTION TANH(X)
00498     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00499     TANH = DTANH(X)
00500 C     PRINT*, ' X =', X, ' TANH(X) =', TANH
00501     RETURN
00502     END
```

(FLECS VERSION 22.51)

```
00503 C
00504     FUNCTION FTANH(THETA)
00505     IMPLICIT DOUBLE PRECISION(A-H,P-Z)
00506     EXTERNAL TANH
00507     COMMON N, AK
00508     COST = DCOS(THETA)
00509     COSNT = DCOS(N*THETA)
00510     FTANH = COSNT
00511     IF (COST.EQ.0) RETURN
```

00512 FTANH = (TANH(AK*COST)/AK)*(COSNT/COST)
00513 RETURN
00514 END

(FLECS VERSION 22.51)

EOI. 0 FILES. 1 RECS. 2321 WORDS.
/

COPY, COMPIL

```

FUNCTION ROMBRG(A, B, F, OE)                                00008
IMPLICIT DOUBLE PRECISION(A-H,P-Z)                        00009
INTEGER FOUR(21), 0                                       00011
DIMENSION Q(20), Q0(2)                                     00012
EQUIVALENCE (Q0(2),Q(1))                                  00013
DATA 0/0/, LMAX/1/, FOUR(1)/4/                            00015
Q(0) = 0                                                   00017
H = (B - A)/2                                             00018
IMAX = 1                                                  00019
DO 99999 L=1,20                                           00021
IF(.NOT.(L.GE.LMAX)) GO TO 99998                          00022
LMAX = L + 1                                              00023
FOUR(L+1) = 4*FOUR(L)                                     00024
99998 SUM = 0                                              00027
DO 99997 I=1,IMAX                                         00028
V = (2*I - IMAX - 1.0D0)/IMAX                             00029
U = V*(3 - V*V)/2                                         00030
UP = 1.5D0*(1 - V*V)                                       00031
X = A + (U + 1)*H                                         00032
SUM = SUM + F(X)*UP                                       00033
99997 CONTINUE                                             00034
SUM = SUM + IMAX*Q(L-1)                                    00035
IMAX = 2*IMAX                                             00036
Q(L) = SUM/IMAX                                           00037
DO 99996 M=1,L                                           00039
K = L - M                                                 00040
Q(K) = (FOUR(M)*Q(K+1) - Q(K))/(FOUR(M) - 1)           00041
99996 CONTINUE                                             00042
IF(.NOT.(L.GT.1)) GO TO 99995                             00044
ROMBRG = (B - A)*Q(0)                                     00045
ODQ = DABS((Q(0) - Q(1))*(B - A))                         00050
IF (ODQ.LE.OE) RETURN                                     00051
99995 CONTINUE                                             00054
99999 CONTINUE                                             00054
RETURN                                                    00055
END                                                        00056
FUNCTION ROMBER(A, B, F, OE)                                00063
IMPLICIT DOUBLE PRECISION(A-H,P-Z)                        00064
INTEGER FOUR(21), 0                                       00066
DIMENSION Q(20), Q0(2)                                     00067
EQUIVALENCE (Q0(2),Q(1))                                  00068
DATA 0/0/, LMAX/1/, FOUR(1)/4/                            00070
Q(0) = (F(A) + F(B))/2                                     00072
H = B - A                                                 00073
IMAX = 1                                                  00074
DO 99999 L=1,20                                           00076
IF(.NOT.(L.GE.LMAX)) GO TO 99998                          00077
LMAX = L + 1                                              00078
FOUR(L+1) = 4*FOUR(L)                                     00079
99998 H = H/2                                             00082
SUM = 0                                                  00083
DO 99997 I=1,IMAX                                         00084
SUM = SUM + F(A + H*(2*I - 1))                            00084
99997 CONTINUE                                             00084
SUM = SUM + IMAX*Q(L-1)                                    00085
IMAX = 2*IMAX                                             00086
Q(L) = SUM/IMAX                                           00087
DO 99996 M=1,L                                           00089
K = L - M                                                 00090
Q(K) = (FOUR(M)*Q(K+1) - Q(K))/(FOUR(M) - 1)           00091

```

99996	CONTINUE	00092
	ROMBER = (B - A)*Q(0)	00094
	ODQ = DABS((Q(0) - Q(1))*(B - A))	00099
	IF (ODQ.LE.DE) RETURN	00100
99999	CONTINUE	00102
	RETURN	00103
	END	00104
	FUNCTION SIN(X)	00107
	IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00108
	SIN = DSIN(X)	00109
	RETURN	00111
	END	00112
	FUNCTION TAN(X)	00114
	IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00115
	TAN = DSIN(X)/DCOS(X)	00116
	RETURN	00118
	END	00119
	FUNCTION ASIN(X)	00121
	IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00122
	ASIN = DATAN(X/DSQRT(1-X*X))	00123
	RETURN	00125
	END	00126
	FUNCTION ACOS(X)	00128
	IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00129
	ACOS = DATAN(DSQRT(1-X*X)/X)	00130
	RETURN	00132
	END	00133
	FUNCTION EXP(X)	00135
	IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00136
	EXP = DEXP(X)	00137
	RETURN	00139
	END	00140
	FUNCTION SQRT(X)	00142
	IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00143
	SQRT = DSQRT(X)	00144
	RETURN	00146
	END	00147
	FUNCTION ATAN(X)	00149
	IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00150
	ATAN = DATAN(X)	00151
	RETURN	00153
	END	00154
	FUNCTION ALOG(X)	00156
	IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00157
	ALOG = DLOG(X)	00158
	RETURN	00160
	END	00161
	FUNCTION COS(X)	00163
	IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00164
	COS = DCOS(X)	00165
	RETURN	00167
	END	00168
	FUNCTION FCOST(THETA)	00170
	IMPLICIT DOUBLE PRECISION (A-H,P-Z)	00171
	COMMON N, AK	00172
	COST = DCOS(THETA)	00173
	COSNT = DCOS(N*THETA)	00174
	FCOST = DCOS(AK*COST)*COSNT	00175
	RETURN	00176
	END	00177
	FUNCTION FSINT(THETA)	00179

IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00180
COMMON N, AK	00181
COST = DCOS(THETA)	00182
COSNT = DCOS(N*THETA)	00183
FSINT = COSNT	00184
IF (COST.EQ.0) RETURN	00185
FSINT = (DSIN(AK*COST)/AK)*(COSNT/COST)	00186
RETURN	00187
END	00188
FUNCTION POLY(X, A, N)	00191
IMPLICIT DOUBLE PRECISION (A-H,P-Z)	00192
DIMENSION A(25)	00193
PROD = A(N)	00194
DO 99999 J=1,N	00195
PROD = PROD*X + A(N-J)	00195
99999 CONTINUE	00195
POLY = PROD	00196
RETURN	00197
END	00198
SUBROUTINE ACOEF(A, B, IC, AK, N)	00200
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00201
DIMENSION B(25), A(25), IC(25,25)	00202
DO 99999 J=1,N	00204
SUM = 0	00205
DO 99998 K=J,N	00206
SUM =SUM + B(K)*IC(K, J)	00206
99998 CONTINUE	00206
A(J) = SUM/AK**J	00207
99999 CONTINUE	00207
J = 0	00210
K = 0	00211
SUM = B(K)*IC(K, J)	00211
DO 99997 K=1,N	00212
SUM = SUM + B(K)*IC(K, J)	00213
99997 CONTINUE	00214
A(J) = SUM	00214
RETURN	00216
END	00217
SUBROUTINE TCH(IC, N)	00218
DIMENSION IC(25, 25)	00219
INTEGER U	00221
DATA 0,0,	00222
IC(0, 0) = 1	00223
IC(0-1, 0) = 0	00225
DO 99999 K=1-N	00226
IC(K, 0) = -IC(K-2, 0)	00227
IC(K-2, K) = 0	00228
IC(0-1, 1) = 1	00228
DO 99998 J=1-N	00230
IF(.NOT.(MOD(J+K,2).EQ.1)) GO TO 99996	00231
IC(J, J) = 0	00231
GO TO 99997	00231
99996 IC(K, J) = 2*IC(K-1, J-1) - IC(K-2, J)	00232
99997 CONTINUE	00234
99998 CONTINUE	00234
99999 CONTINUE	00235
RETURN	00236
END	00237
FUNCTION FTANT(THETA)	00239
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00240
EXTERNAL	00241

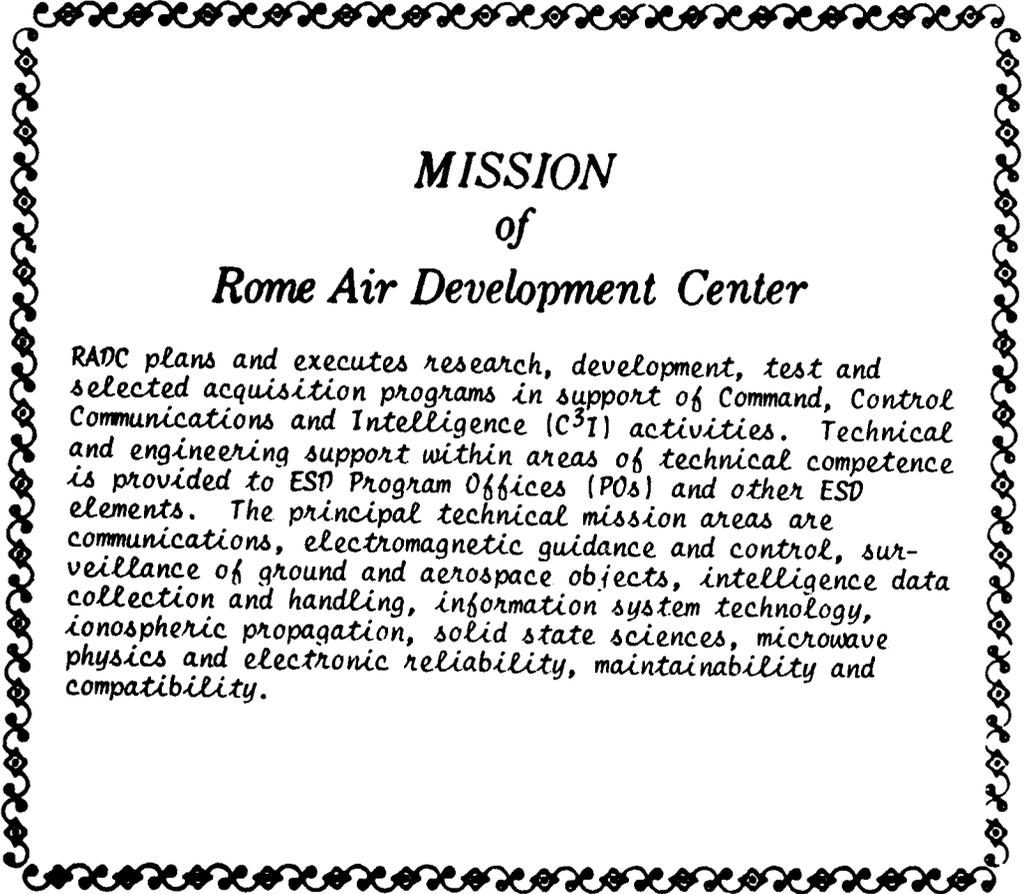
COMMON N, AK	00242
COST = DCOS(THETA)	00243
COSNT = DCOS(N*THETA)	00244
FINT = COSNT	00245
IF (COST.EQ.0) RETURN	00246
FINT = (TAN(AK*COST)/AK)*(COSNT/COST)	00247
RETURN	00248
END	00249
FUNCTION FATANT(THETA)	00251
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00252
COMMON N, AK	00253
COST = DCOS(THETA)	00254
COSNT = DCOS(N*THETA)	00255
FATANT = COSNT	00256
IF (COST.EQ.0) RETURN	00257
FATANT = (ATAN(AK*COST)/AK)*(COSNT/COST)	00258
RETURN	00259
END	00260
FUNCTION FASINT(THETA)	00262
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00263
EXTERNAL ASIN	00264
COMMON N, AK	00265
COST = DCOS(THETA)	00266
COSNT = DCOS(N*THETA)	00267
FASINT = COSNT	00268
IF (COST.EQ.0) RETURN	00269
FASINT = (ASIN(AK*COST)/AK)*(COSNT/COST)	00270
RETURN	00271
END	00272
FUNCTION FACOST(THETA)	00274
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00275
EXTERNAL ACOS	00276
COMMON N, AK	00277
COST = DCOS(THETA)	00278
COSNT = DCOS(N*THETA)	00279
FACOST = ACOS(AK*COST)*COSNT	00280
RETURN	00281
END	00282
FUNCTION RASIN(X)	00284
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00285
EXTERNAL ASIN	00286
RASIN = 1/ASIN(X)	00287
RETURN	00289
END	00290
FUNCTION FRASINT(THETA)	00292
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00293
EXTERNAL RASIN	00294
COMMON N, AK	00295
COST = DCOS(THETA)	00296
COSNT = DCOS(N*THETA)	00297
FRASINT = COSNT	00298
IF (COST.EQ.0) RETURN	00299
FRASINT = COST*RASIN(AK*COST)*AK*COSNT	00300
RETURN	00301
END	00302
FUNCTION FALNT(THETA)	00304
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00305
EXTERNAL ALN	00306
COMMON N, AK	00307
COST = DCOS(THETA)	00308
COSNT = DCOS(N*THETA)	00309

FALNT = COSNT	00310
IF (COST.EQ.0) RETURN	00311
FALNT = (ALN(AK*COST)/AK)*(COSNT/COST)	00312
RETURN	00313
END	00314
FUNCTION ALN(Z)	00316
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00317
ALN = DLOG((Z + 1)/(1 - Z))/2	00318
RETURN	00320
END	00321
FUNCTION FLOGT(THETA)	00323
IMPLICIT DOUBLE PRECISION (A-H,P-Z)	00324
COMMON N, AK, I, AL	00325
COST = DCOS(THETA)	00326
COSNT = DCOS(N*THETA)	00327
FLOGT = DLOG((AK + AL + (AK - AL)*COST)/2)*COSNT	00328
RETURN	00329
END	00330
FUNCTION FSQRTT(THETA)	00332
IMPLICIT DOUBLE PRECISION (A-H,P-Z)	00333
COMMON N, AK, I, AL	00334
COST = DCOS(THETA)	00335
COSNT = DCOS(N*THETA)	00336
FSQRTT = DSQRT((AK + AL + (AK - AL)*COST)/2)*COSNT	00337
RETURN	00338
END	00339
FUNCTION FEXPT(THETA)	00341
IMPLICIT DOUBLE PRECISION (A-H,P-Z)	00342
COMMON N, AK, I, AL	00343
COST = DCOS(THETA)	00344
COSNT = DCOS(N*THETA)	00345
FEXPT = DEXP((AK + AL + (AK - AL)*COST)/2)*COSNT	00346
RETURN	00347
END	00348
FUNCTION FAASNT(THETA)	00350
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00351
EXTERNAL ASIN	00352
COMMON N, AK, I, AL	00353
COST = DCOS(THETA)	00354
COSNT = DCOS(N*THETA)	00355
FAASNT = ASIN((AK + AL + (AK - AL)*COST)/2)*COSNT	00356
RETURN	00357
END	00358
FUNCTION EX1X(X)	00360
IMPLICIT DOUBLE PRECISION (A-H,P-Z)	00361
EX1X = 1	00362
IF (X.NE.0) EX1X = (DEXP(X) - 1)/X	00363
RETURN	00365
END	00366
FUNCTION FEX1XT(THETA)	00368
IMPLICIT DOUBLE PRECISION (A-H,P-Z)	00369
COMMON N, AK, I, AL	00370
COST = DCOS(THETA)	00371
COSNT = DCOS(N*THETA)	00372
FEX1XT = EX1X((AK + AL + (AK - AL)*COST)/2)*COSNT	00373
RETURN	00374
END	00375
FUNCTION SQAC(Y)	00377
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00379
SQAC = 1	00381
IF (Y.EQ.0) RETURN	00382

SQAC = DATAN(DSQRT(Y*(2 - Y))/(1 - Y))**2/(2*Y)	00384
RETURN	00386
END	00387
FUNCTION FSQACT(THETA)	00389
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00390
COMMON N, AK, I, AL	00391
COST = DCOS(THETA)	00392
COSNT = DCOS(N*THETA)	00393
FSQACT = SQAC((AK + AL + (AK - AL)*COST)/2)*COSNT	00394
RETURN	00395
END	00396
FUNCTION SINHX)	00398
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00399
E = DEXP(X)	00400
SINH = (E - 1/E)/2	00401
RETURN	00403
END	00404
FUNCTION COSHX)	00405
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00407
E = DEXP(X)	00408
COSH = (E + 1/E)/2	00409
RETURN	00411
END	00412
FUNCTION FSINHT(THETA)	00414
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00415
EXTERNAL SINH	00416
COMMON N, AK	00417
COST = DCOS(THETA)	00418
COSNT = DCOS(N*THETA)	00419
FSINHT = COSNT	00420
IF (COST.EQ.0) RETURN	00421
FSINHT = (SINH(AK*COST)/AK)*(COSNT/COST)	00422
RETURN	00423
END	00424
FUNCTION FCOSHT(THETA)	00426
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00427
EXTERNAL COSH	00428
COMMON N, AK	00429
COST = DCOS(THETA)	00430
COSNT = DCOS(N*THETA)	00431
FCOSHT = COSH(AK*COST)*COSNT	00432
RETURN	00433
END	00434
FUNCTION ASINHX)	00436
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00437
EXTERNAL ATANH	00438
ASINH = 0.0D0	00439
IF (X .EQ. 0) RETURN	00440
ASINH = ATANH(X/DSQRT(1 + X*X))	00441
RETURN	00443
END	00444
FUNCTION FASINHT(THETA)	00446
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00447
EXTERNAL ASINH	00448
COMMON N, AK	00449
COST = DCOS(THETA)	00450
COSNT = DCOS(N*THETA)	00451
FASINHT = COSNT	00452
IF (COST.EQ.0) RETURN	00453
FASINHT = (ASINH(AK*COST)/AK)*(COSNT/COST)	00454
RETURN	00455

END	00456
FUNCTION ACOSH(X)	00458
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00459
EXTERNAL ATANH	00460
ACOSH = ATANH(DSQRT(X*X-1))/X	00461
RETURN	00463
END	00464
FUNCTION FACOSH(THETA)	00466
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00467
EXTERNAL ACOSH	00468
COMMON N, AN	00469
COSI = DCOS(THETA)	00470
COSN = DCOSN(THETA)	00471
ACOSH = ACOSH(AN*(COST+COSNT))	00472
RETURN	00473
END	00474
FUNCTION ATANH(X)	00476
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00477
ATANH = 0.5*LOG((1+X)/(1-X))	00478
IF X.LE.0, GO TO RETURN	00479
ATANH = 0.5*LOG((1+X)/(1-X))	00480
RETURN	00482
END	00483
FUNCTION FATANH(THETA)	00485
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00486
EXTERNAL ATANH	00487
COMMON N, AN	00488
COSI = DCOS(THETA)	00489
COSN = DCOSN(THETA)	00490
FATANH = 0.5*ATANH(AN*(COST+COSNT))	00491
IF X.LE.0, GO TO RETURN	00492
FATANH = 0.5*ATANH(AN*(COST+COSNT))	00493
RETURN	00494
END	00495
FUNCTION ATANH(X)	00497
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00498
TANH = ATANH(X)	00499
RETURN	00501
END	00502
FUNCTION FTANH(THETA)	00504
IMPLICIT DOUBLE PRECISION(A-H,P-Z)	00505
EXTERNAL TANH	00506
COMMON N, AN	00507
COSI = DCOS(THETA)	00508
COSN = DCOSN(THETA)	00509
FTANH = TANH(AN*(COST+COSNT))	00510
IF X.LE.0, GO TO RETURN	00511
FTANH = TANH(AN*(COST+COSNT))	00512
RETURN	00513
END	00514

END OF FILE. 1 RECS. 3735 WORDS.

A decorative border with a repeating floral or scrollwork pattern surrounds the central text.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C³I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

FILMED
7-8